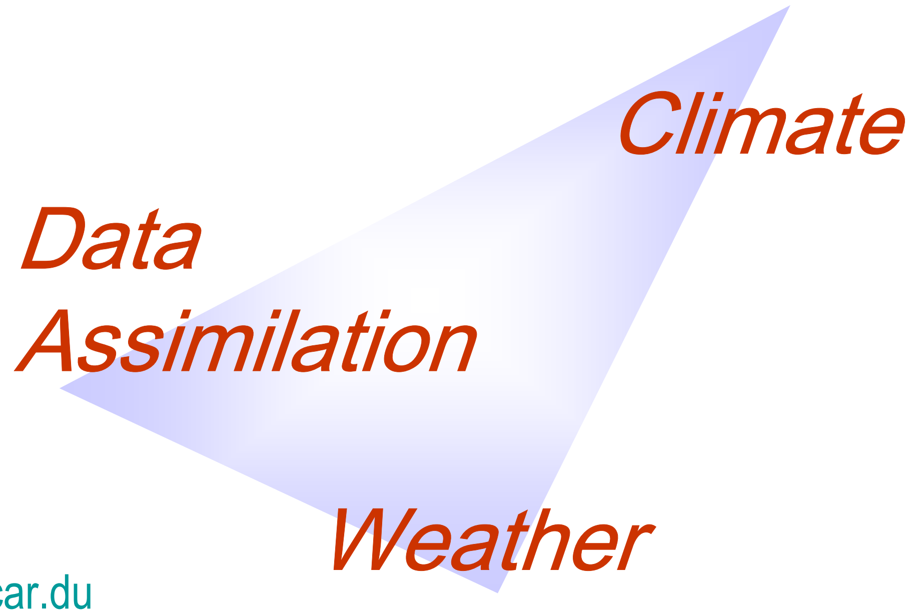
NOTICE:

Distribution authorized to the Department of Defense and U.S. DoD contractors only. Other requests shall be referred to: Contracting Officer, PET II, Code N111

Stennis Space Center, MS

Introduction to the Earth System Modeling Framework



Cecelia DeLuca cdeluca@ucar.du

Nancy Collins nancy@ucar.edu

Jon Wolfe jwolfe@ucar.edu

January 18-19, 2005

Goals of this Tutorial

1. To give future ESMF users an understanding of the background, goals, and scope of the ESMF project
2. To review the status of the ESMF software implementation and current application adoption efforts
3. To outline the overall design and principles underlying the ESMF software
4. To describe the major classes and functions of ESMF in sufficient detail to give future users an understanding of how ESMF could be utilized in their own codes
5. To describe in steps how a user code prepares for using ESMF, incorporates ESMF, and runs under ESMF
6. To identify ESMF resources available to users such as documentation, mailing lists, and support staff
7. To define what is required for ESMF compliance
8. To examine and work with code examples in order to demonstrate ESMF adoption and use

Specific Topics

- Standard behaviors and interfaces across ESMF
- Bottom-up and top-down approaches to adoption
- What it means to become an ESMF Component
- Defining hierarchical applications with Gridded Components and Coupler Components
- Creating and manipulating State, Field and Grid classes
- Setting up applications for sequential or concurrent execution
- Why there is an ESMF Virtual Machine
- How to use ESMF utilities such as Time Manager, LogErr, and Configuration Attributes

ESMF Website

http://www.esmf.ucar.edu

See this site for downloads, documentation, references, repositories, meeting schedules, test archives, and just about anything else you need to know about ESMF.

References to ESMF documentation in this tutorial correspond to the documentation releases with ESMF Version 2.1.0.

1 BACKGROUND, GOALS, AND SCOPE

- Overview
- ESMF and the Community
- The ESMF Organization
- Goals and Rationale for Adoption
- Exercises

Motivation

In climate research and NWP...

increased emphasis on detailed representation of individual physical processes; requires many teams of specialists to contribute components to an overall modeling system

In computing technology...

increase in hardware and software complexity in high-performance computing, as we shift toward the use of scalable computing architectures

In software …

development of frameworks, such as FMS, GEMS, CCA and WRF, that encourage software reuse and interoperability

The ESMF is a focused community effort to tame the complexity of models and the computing environment. It leverages, unifies and extends existing software frameworks, creating new opportunities for scientific contribution and collaboration.

Background

NASA's Earth Science Technology Office proposed the creation of an Earth System Modeling Framework (ESMF) in the September 2000 NASA Cooperative Agreement Notice (CAN):

> "Increasing Interoperability and Performance of Grand Challenge Applications in the Earth, Space, Life and Microgravity Sciences"

A large, interagency collaboration with roots in the Common Modeling Infrastructure Working Group proposed three interlinked projects to develop and deploy the ESMF, which were all funded:

Part I: Core ESMF Development (PI: Killeen, NCAR)
Part II: Modeling Applications (PI: Marshall, MIT)
Part III: Data Assimilation Applications (PI: da Silva, NASA GMAO)

NASA CAN ESMF Project Description

GOALS: To increase software reuse, interoperability, ease of use and performance portability in climate, weather, and data assimilation applications

PRODUCTS:

- Core framework: Software for coupling geophysical components and utilities for building components
- Applications: Deployment of the ESMF in 15 of the nation's leading climate and weather models, assembly of 8 new science-motivated applications

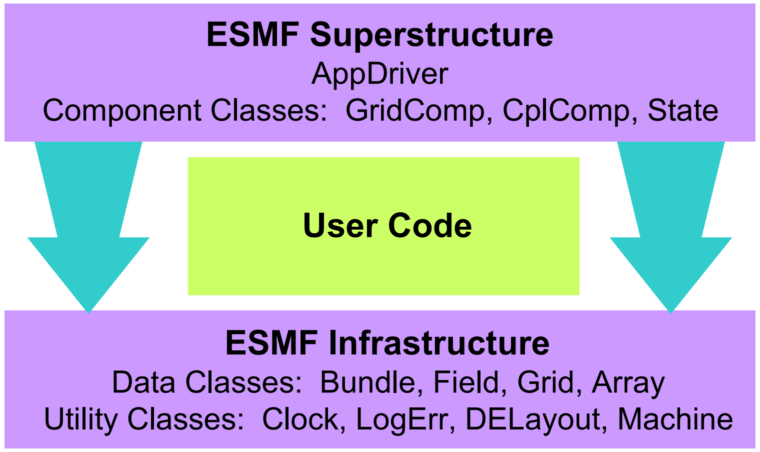
METRICS:

Reuse	Interoperability	Ease of Adoption	Performance
15 applications use ESMF component coupling services and 3+ utilities	8 new applications comprised of never-before coupled components	2 codes adopt ESMF with < 2% lines of code changed, or within 120 FTE-hours	No more than 10% overhead in time to solution, no degradation in scaling

RESOURCES and TIMELINE: $9.8M over 3 years

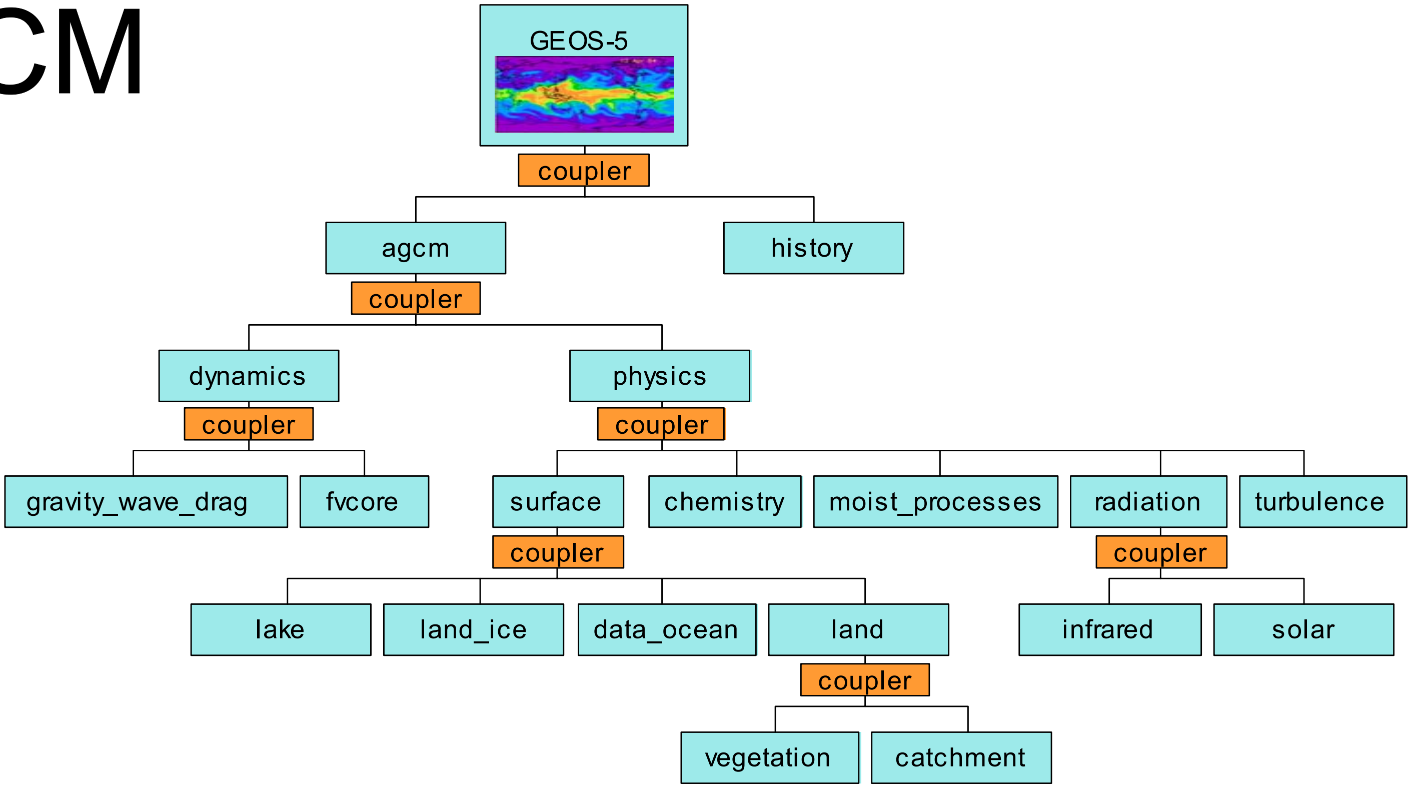
What is ESMF?

1. ESMF provides tools for turning model codes into components with standard interfaces and standard drivers
2. ESMF provides data structures and common utilities that components use
   i. to organize codes
   ii. to improve performance portability
   iii. for common services such as data communications, regridding, time management and message logging



ESMF Superstructure
AppDriver
Component Classes: GridComp, CplComp, State
User Code
ESMF Infrastructure
Data Classes: Bundle, Field, Grid, Array
Utility Classes: Clock, LogErr, DELayout, Machine

Application Example: GEOS-5 AGCM



- Each box is an ESMF component
- Every component has a standard interface so that it is swappable
- New components can easily be added to the hierarchical system
- Data in and out of components are packaged as state types
- Coupling tools include regridding and redistribution methods

1 BACKGROUND, GOALS, AND SCOPE

- Overview
- ESMF and the Community
- The ESMF Organization
- Goals and Rationale for Adoption
- Exercises

ESMF is a Community Effort

- Collaborators and customers include:
  - NSF NCAR
  - NOAA GFDL, NOAA NCEP
  - DOE LANL, DOE ANL
  - NASA GMAO, NASA Land Information Systems, NASA GISS
  - DoD Navy, Air Force, and Army (new)
  - University of Michigan, UCLA, MIT
- Users define development priorities
- Users actively test and evaluate the framework design and implementation
- ~15% of ESMF source code is from user contributions (IO from WRF, resource file manager from GMAO, regridding from Los Alamos)

Open Source Development

- Open source license (GPL)
- Open source environment (SourceForge)
- Open repositories: web-browsable CVS repositories accessible from the ESMF website
    - for source code
    - for contributions (currently porting contributions and performance testing)
- Open development priorities and schedule: priorities set based on user meetings, telecons, and mailing list discussions, web-browsable task lists
- Open testing: 1000+ tests are bundled with the ESMF distribution and can be run by users
- Open port status: results of nightly tests on many platforms are web-browsable
- Open metrics: test coverage, lines of code, requirements status are updated regularly and are web-browsable

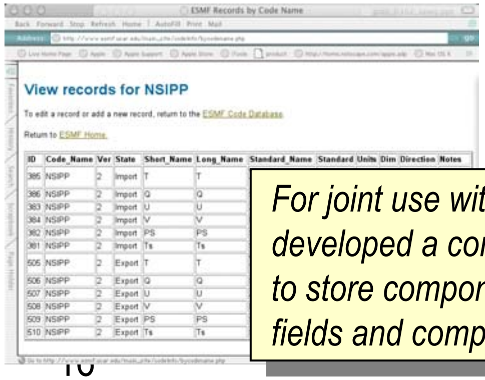
Open Source Constraints

- ESMF does not allow unmoderated check-ins to its main source CVS repository (though there is minimal check-in oversight for the contributions repository)
- ESMF has a co-located, line managed Core Team whose members are dedicated to framework implementation and support – it does not rely on volunteer labor
- ESMF actively sets priorities based on user needs and feedback
- ESMF requires that contributions follow project conventions and standards for code and documentation
- ESMF schedules regular releases and meetings

The above are necessary for development to proceed at the pace desired by sponsors and users, and to provide the level of quality and customer support necessary for codes in this domain

Related Projects

- PRISM is an ongoing European Earth system modeling infrastructure project
- Involves current state-of-the-art atmosphere, ocean, sea-ice, atmospheric chemistry, land-surface and ocean-biogeochemistry models
- 22 partners: leading climate researchers and computer vendors, includes MPI, KNMI, UK Met Office, CERFACS, ECMWF, DMI
- ESMF is working with PRISM to merge frameworks and develop common conventions



ESMF Records by Code Name

View records for NSIPP

To edit a record or add a new record, return to the ESMF Code Database

Return to ESMF Home

ID	Code_Name	Ver	State	Short_Name	Long_Name	Standard_Name	Standard Units	Dim	Direction	Notes
385	NSIPP	2	Import	T	T					
386	NSIPP	2	Import	Q	Q					
383	NSIPP	2	Import	U	U					
384	NSIPP	2	Import	V	V					
382	NSIPP	2	Import	PS	PS					
381	NSIPP	2	Import	Ts	Ts					
505	NSIPP	2	Export	T	T					
506	NSIPP	2	Export	Q	Q					
507	NSIPP	2	Export	U	U					
508	NSIPP	2	Export	V	V					
509	NSIPP	2	Export	PS	PS					
510	NSIPP	2	Export	Ts	Ts					

For joint use with PRISM, ESMF developed a component database to store component import/export fields and component descriptions

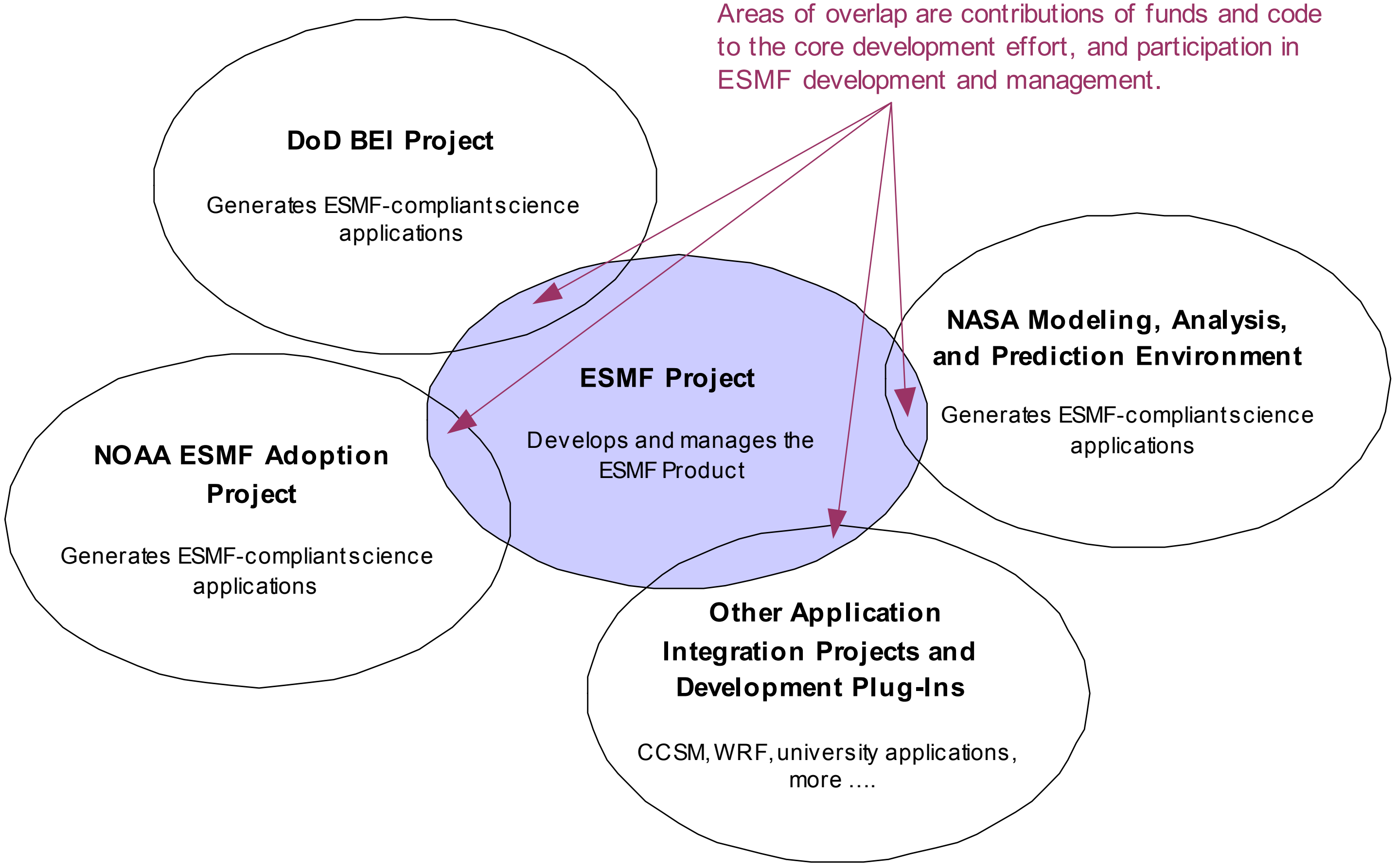
- CCA is creating a minimal interface and sets of tools for linking high performance components. CCA can be used to implement frameworks and standards developed in specific domains (such as ESMF).
- Collaborators include LANL, ANL, LLNL, ORNL, Sandia, University of Tennessee, and many more. Ongoing ESMF collaboration with CCA/LANL on language interoperability.
- Working prototype demonstrating CCA/ESMF interoperability, to be presented at SC2003.

1 BACKGROUND, GOALS, AND SCOPE

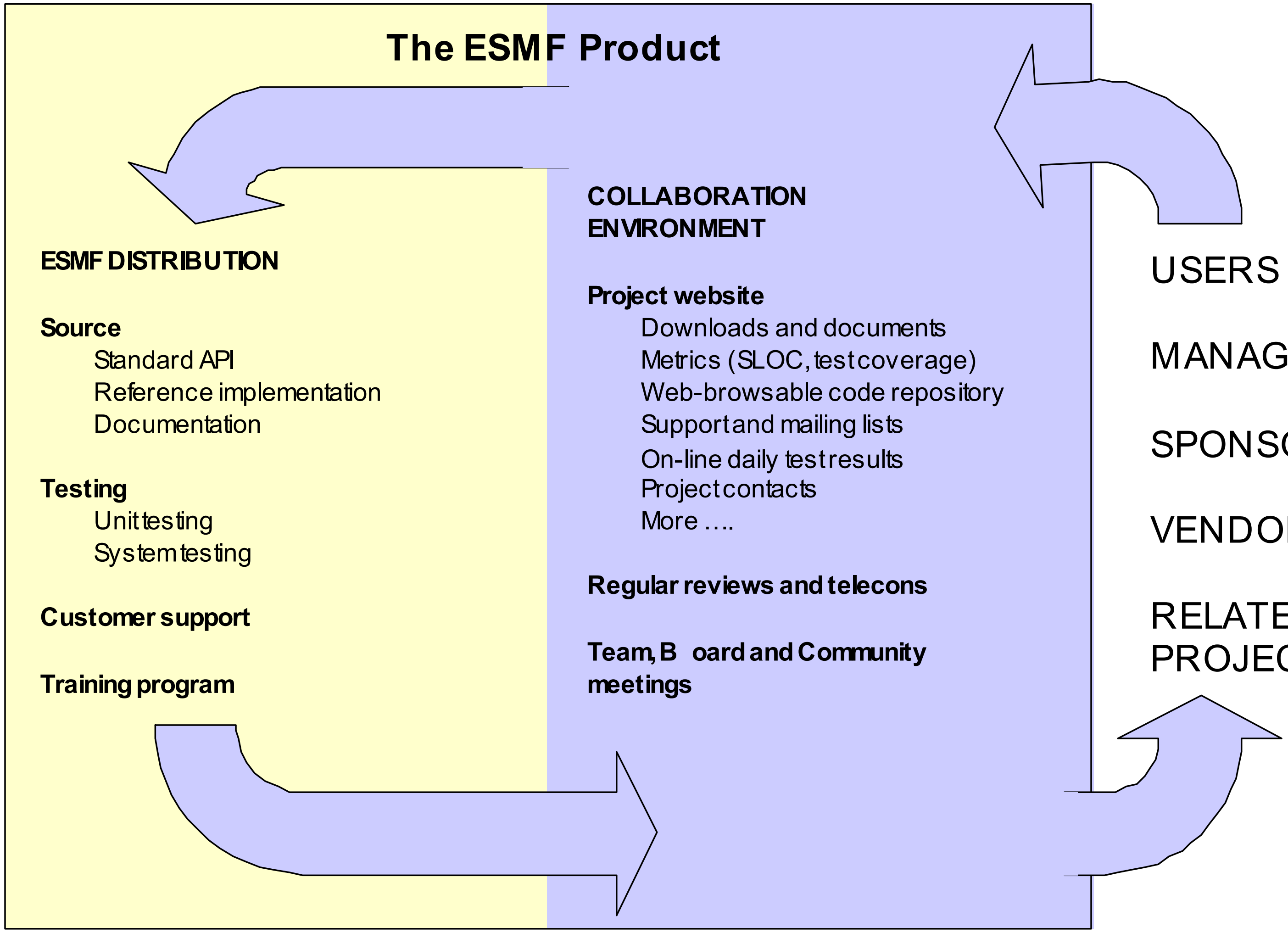
- Overview
- ESMF and the Community
- The ESMF Organization
- Goals and Rationale for Adoption
- Exercises

ESMF and Application Integration Projects

Application Integration Projects and Development Plug-Ins from the DoD, NASA, NOAA, and elsewhere interact with the ESMF Project.
Areas of overlap are contributions of funds and code to the core development effort, and participation in ESMF development and management.


DoD BEI Project
Generates ESMF-compliant science applications
ESMF Project
Develops and manages the ESMF Product
NASA Modeling, Analysis, and Prediction Environment
Generates ESMF-compliant science applications
NOAA ESMF Adoption Project
Generates ESMF-compliant science applications
Other Application Integration Projects and Development Plug-Ins
CCSM, WRF, university applications, more ….

The ESMF Product



The ESMF Product

ESMF DISTRIBUTION

Source
- Standard API
- Reference implementation
- Documentation

Testing
- Unit testing
- System testing

Customer support

Training program

COLLABORATION ENVIRONMENT

Project website
- Downloads and documents
- Metrics (SLOC, test coverage)
- Web-browsable code repository
- Support and mailing lists
- On-line daily test results
- Project contacts
- More ….

Regular reviews and telecons

Team, Board and Community meetings

USERS

MANAGEMENT

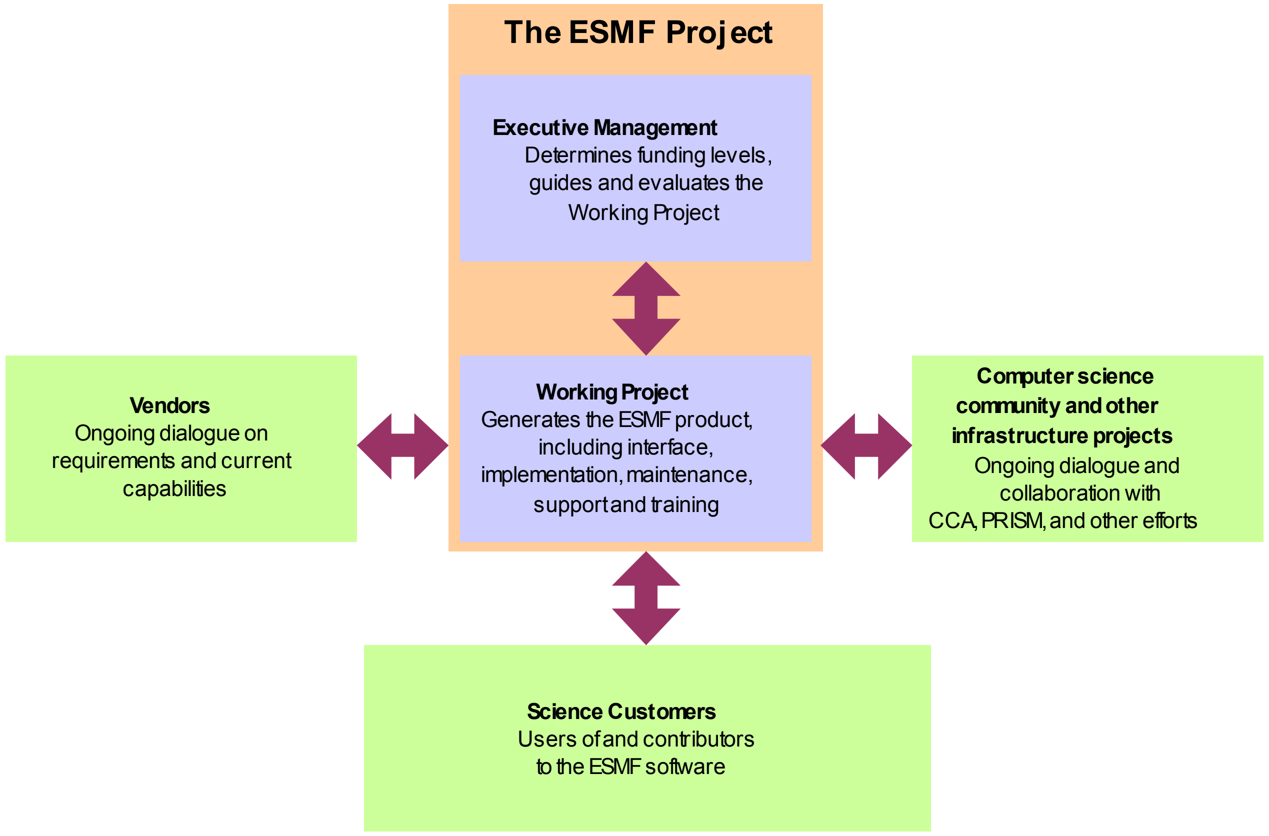
SPONSORS

VENDORS

RELATED PROJECTS

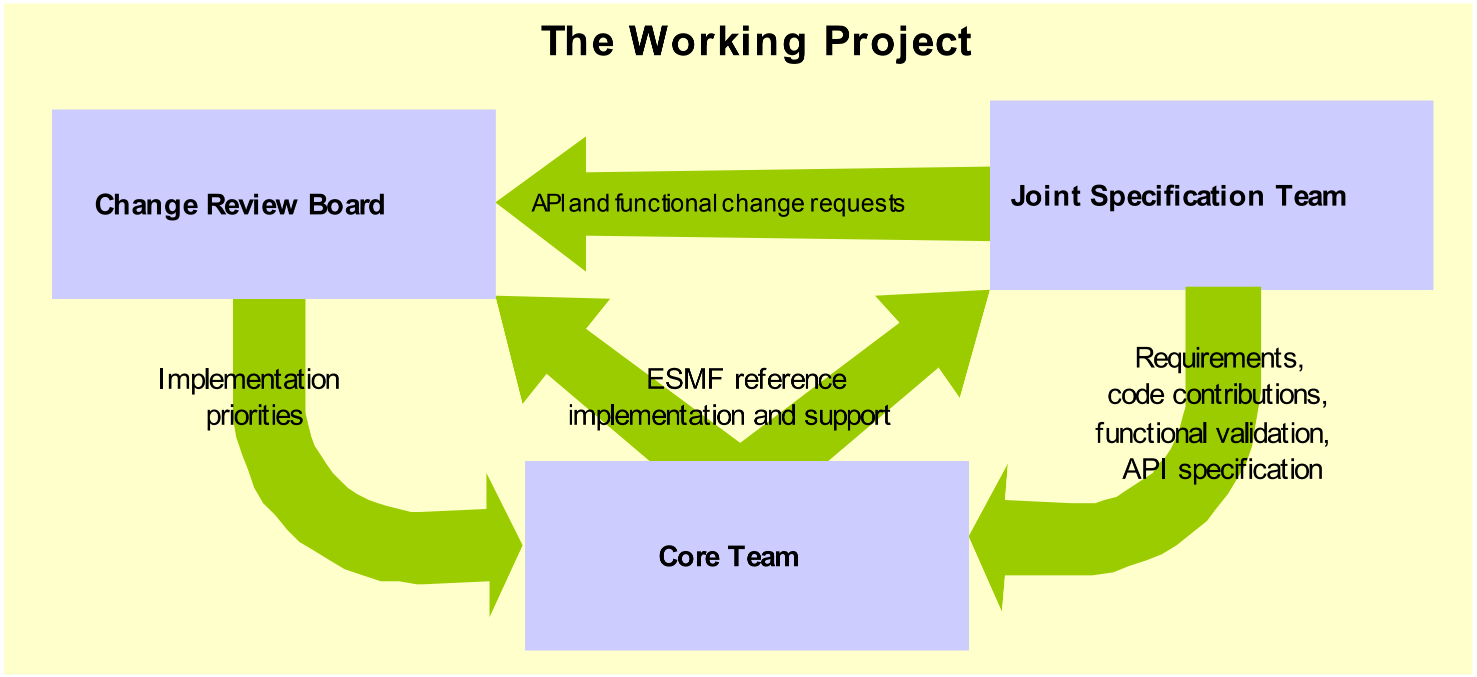
The ESMF Project

- The ESMF Project is responsible for directing and delivering the ESMF Product.
- The organization is designed to encourage collaboration at all levels: hands on developer/user, institutional director, agency


The ESMF Project
Executive Management
Determines funding levels, guides and evaluates the Working Project
Working Project
Generates the ESMF product, including interface, implementation, maintenance, support and training
Vendors
Ongoing dialogue on requirements and current capabilities
Computer science community and other infrastructure projects
Ongoing dialogue and collaboration with CCA, PRISM, and other efforts
Science Customers
Users of and contributors to the ESMF software

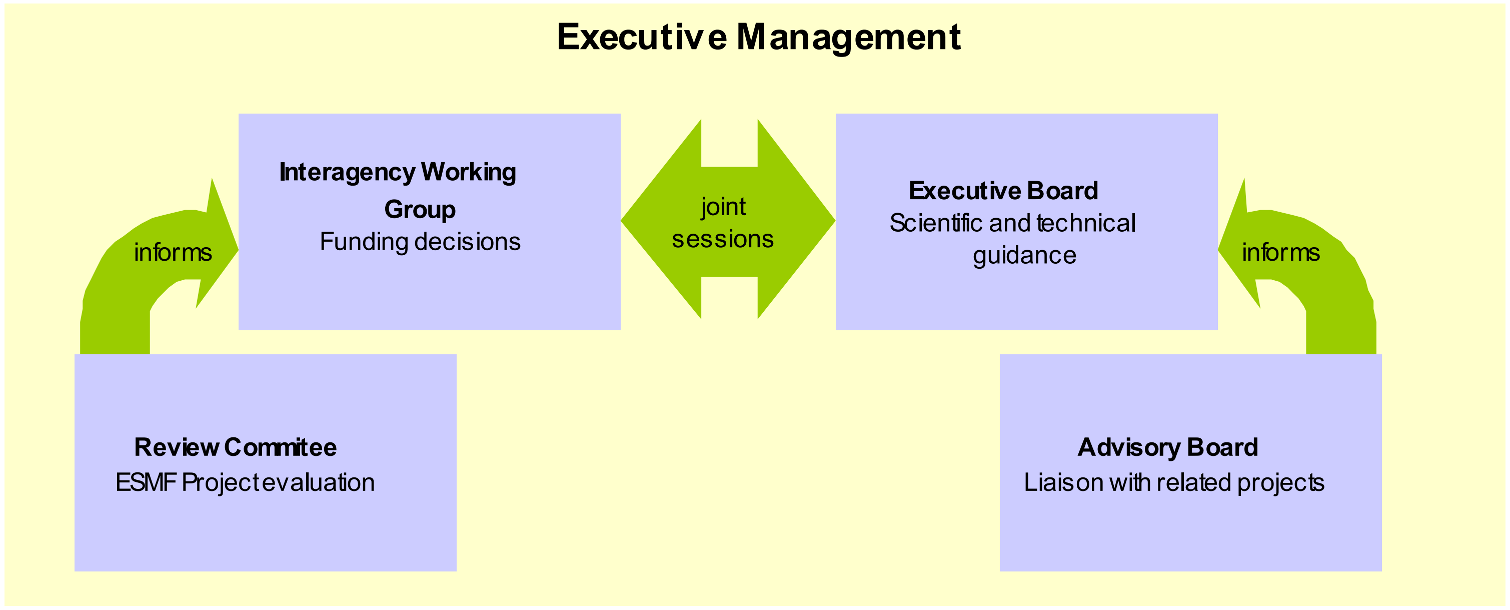
The ESMF Working Project

- Implements the ESMF product day-to-day
- Three parts:
  - Core Team – development and maintenance, support and training, testing, web
  - Joint Specification Team – hands-on users and developers, weekly telecons
  - Change Review Board – priorities and schedules for code changes, newly established


The Working Project
Change Review Board
API and functional change requests
Joint Specification Team
Implementation priorities
ESMF reference implementation and support
Requirements, code contributions, functional validation, API specification
Core Team

Executive Management

- Oversees the project
- Four parts
  - Executive Board – sets overall priorities and direction
  - Advisory Board – guidance and coordination
  - Interagency Working Group – agency executives and sponsors
  - Review Committee - evaluation



More Information

For more on the ESMF organization, see the ESMF Draft Project Plan on the ESMF website:

http://www.esmf.ucar.edu > **Publications & Talks**

1 BACKGROUND, GOALS, AND SCOPE

- Overview
- ESMF and the Community
- The ESMF Organization
- Goals and Rationale for Adoption
- Exercises

ESMF Goals

1. Increase scientific productivity by making modeling and analysis software components much easier to build, combine, and exchange, and by enabling modelers to take full advantage of high-end computers.

2. Unify the national and international Earth system modeling community through a common modeling paradigm and regular interactions at all levels.

Why Should I Adopt ESMF If I Already Have a Working Model?

- There is an emerging pool of other ESMF-based science components that you will be able to interoperate with to create applications - a framework for interoperability is only as valuable as the set of groups that use it, and ESMF has a broad customer base.
- It will reduce the amount of infrastructure code that you need to maintain and write, and allow you to focus more resources on science development.
- ESMF provides solutions to two of the hardest problems in model development: structuring large, multi-component applications so that they are easy to use and extend, and achieving performance portability on a wide variety of parallel architectures.
- It may be better software (better features, better performance portability, better tested, better documented and better funded into the future) than the infrastructure software that you are currently using.
- Community development and use means that the ESMF software is widely reviewed and tested, and that you can leverage contributions from other groups.

1 BACKGROUND, GOALS, AND SCOPE

- Overview
- ESMF and the Community
- The ESMF Organization
- Goals and Rationale for Adoption
- Exercises

Exercises

1. Sketch a diagram of the major components in your application and how they are connected.
2. Introduction of tutorial participants.

Application Diagram

2 STATUS OF DEVELOPMENT AND APPLICATIONS

- Development Status and Priorities
- Performance
- NASA CAN ESMF Project Status
- BEI Codes
- Exercises

ESMF Development Status

- Overall architecture is well-defined and well-accepted
- Components and low-level communications stable
- Logically rectangular grids with regular and arbitrary distributions implemented
- On-line parallel regridding (bilinear, 1st order conservative) completed and optimized
- Other parallel methods, e.g. halo, redistribution, low-level comms implemented
- Utilities such as time manager, logging, and configuration manager usable and adding features
- Virtual machine with uniform interface to shared / distributed memory implemented, hooks for load balancing implemented

ESMF Platform Support

- IBM AIX (32 and 64 bit addressing)
- SGI IRIX64 (32 and 64 bit addressing)
- SGI Altix (64 bit addressing)
- Cray X1 (64 bit addressing)
- Compaq OSF1 (64 bit addressing)
- Linux Intel (32 and 64 bit addressing, with mpich and lam)
- Linux PGI (32 bit addressing, with mpich)
- Linux NAG (32 bit addressing, with mpich)
- Linux Absoft (32 bit addressing, with mpich)
- Linux Lahey (32 bit addressing, with mpich)
- Mac OS X with xlf (32 bit addressing, with lam)

ESMF Distribution Summary

- Fortran interfaces and complete documentation
- Many C++ interfaces, no manuals yet
- Serial or parallel execution (mpiuni stub library)
- Sequential or concurrent execution
- SPMD support

ESMF Near-Term Priorities, FY05

- Concurrent components working on all platforms
- Reworked design and implementation of array / grid / field interfaces and array-level communications
- Optimized wholly irregular grid distributions, regridding and low-level communications
- Grid merges
- Unstructured grids
- Read/write interpolation weights and grid specifications
- Asynchronous I/O
- Support for real time types and other enhancements to utilities

ESMF Longer-Term Priorities

- Improve portability, performance, and error handling, and expand and improve documentation, tutorial materials, and training program
- Develop and assimilate contributions of new functionality into the ESMF software (nested and adaptive grids, data assimilation support including adjoints, load balancing, MPMD, improved IO and utilities)
- Transition the collaboration environment and project organization so that it is effective with multiple sponsors and a larger number of collaborators
- Expand the program of collaboration with CCA, PRISM and other national and international infrastructure initiatives;
- Begin design and implementation of Earth System Modeling Environment (ESME)

ESMF Current Challenges

- Quality and correctness of source code, especially numerical methods
- Process for design and interface review
- Development of advanced grids and regridding
- Requirements database and requirements tracking – new software packages being explored
- Clear, complete, carefully edited documentation and training program materials

Some Metrics …

- Core Team currently has
  - 2 FTE testers,
  - 1/2 FTE performance analyst,
  - 5 FTE developers
  - 1 FTE admin/web support
  - 1 manager
- Test suite currently consists of
  - ~1200 unit tests
  - ~15 system tests,
  - ~35 examples

  runs every night on ~12 platforms
- ~273 ESMF interfaces implemented, ~250 fully or partially tested, ~91% fully or partially tested.
- ~142,000 SLOC, ~26,000 lines of text
- ~63 open bugs, ~316 closed bugs
- ~785 downloads

More Information

For more on scheduling and releases, see the on-line listing:

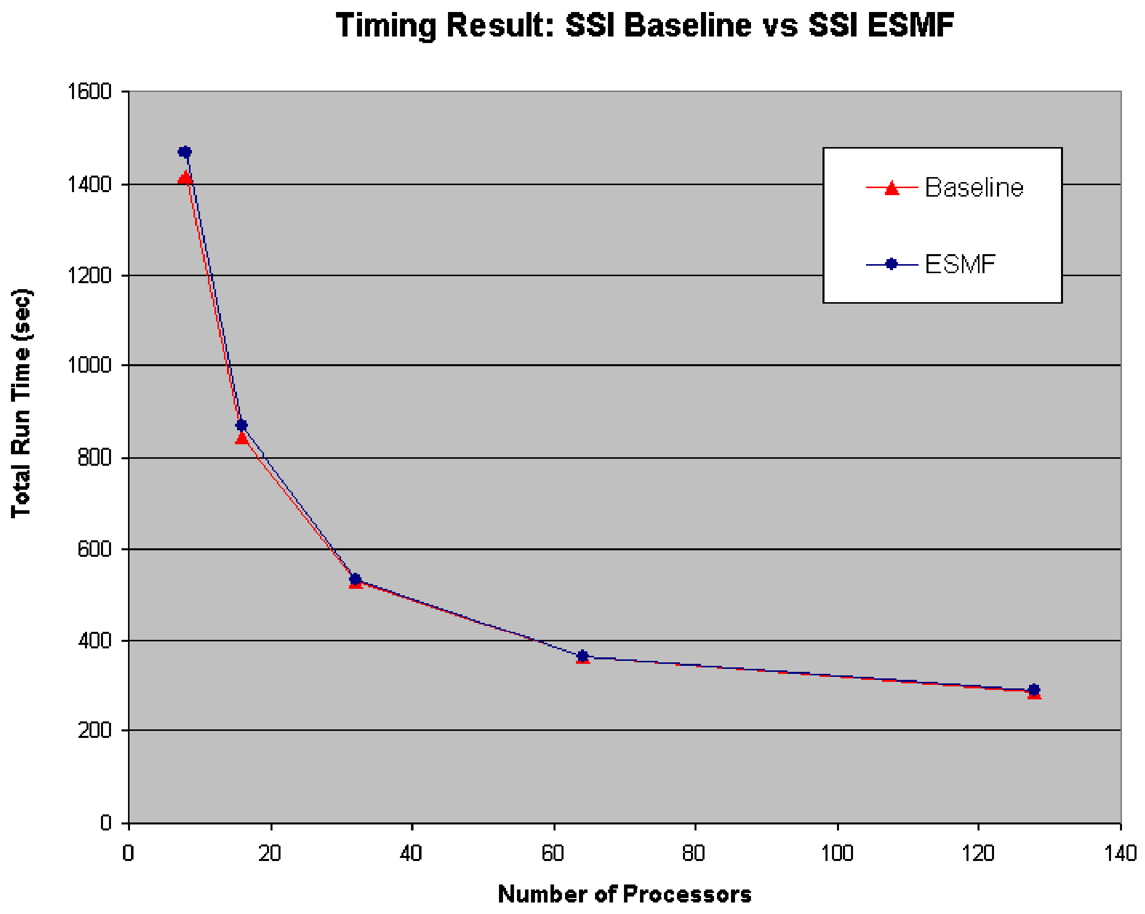
http://www.esmf.ucar.edu > **Development**

Tasks are on the ESMF SourceForge site, under ESMF **Core Tasks**.

2 STATUS OF DEVELOPMENT AND APPLICATIONS

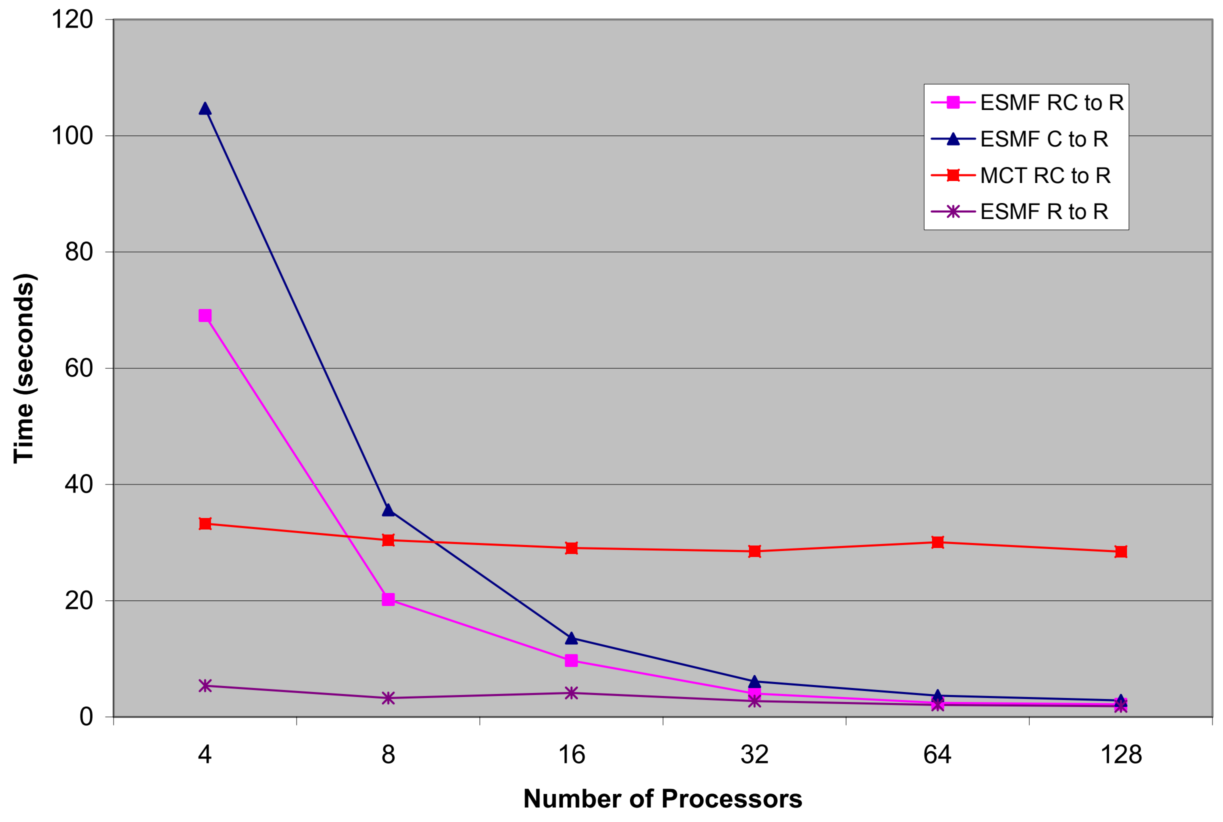
- Development Status and Priorities
- Performance
- NASA CAN ESMF Project Status
- BEI Codes
- Exercises

ESMF Component Overhead



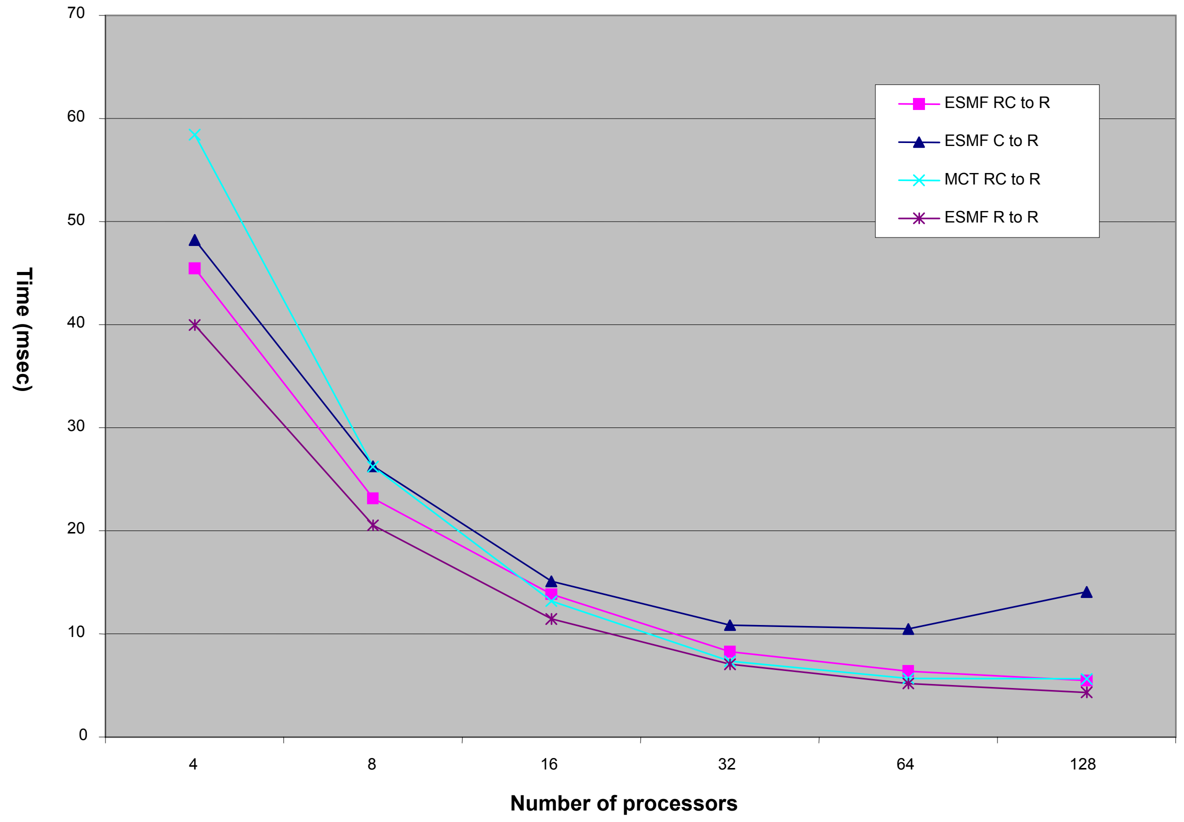
- Measures overhead of ESMF superstructure in NCEP Spectral Statistical Analysis (SSI), ~1% overall
- Run on NCAR IBM
- Runs done by JPL staff, confirmed by NCEP developers

ESMF Regridding Performance, Initialization



- Comparison with the Argonne Model Coupling Toolkit (MCT) bundled with CCSM
- Run on NCAR IBM
- Runs done by JPL staff, not yet confirmed by Argonne developers

ESMF Regridding Performance, Run Time



- Comparison with the Argonne Model Coupling Toolkit (MCT) bundled with CCSM
- Run on NCAR IBM
- Runs done by JPL staff, not yet confirmed by Argonne developers

2 STATUS OF DEVELOPMENT AND APPLICATIONS

- Development Status and Priorities
- Performance
- NASA CAN ESMF Project Status
- BEI Codes
- Exercises

NASA CAN Deliverable Schedule and Metrics

- Public delivery of prototype ESMF v1.0 in May 2003
- Completion of first coupling demonstrations using ESMF in March 2004
- Delivered ESMF v2.0 in June 2004
- Delivery of ESMF v2.1.0 in January 2005 (includes concurrency)
- Delivery of ESMF v2.2.0 anticipated in May 2005
- All project codes scheduled to achieve partial adoption (use of the ESMF component layer and coupling) by November 2004
- All project codes scheduled to achieve full adoption (use of the component layer and coupling plus 3 or more utilities) by June 2005

NASA CAN Modeling Codes

SOURCE	APPLICATION
GFDL	FMS B-grid atmosphere FMS spectral atmosphere FMS MOM4 ocean model
MIT	MITgcm coupled atmosphere/ocean MITgcm regional and global ocean
GMAO	GMAO atmospheric GCM coupled with ocean GCM
NCAR/LANL	CCSM2 including CAM and CLM coupled with POP ocean and data ice model

NASA CAN Data Assimilation Codes

SOURCE	APPLICATION
GMAO	Gridpoint Statistical Interpolation (GSI) System (joint with NCEP) *replaces Physical-space Statistical Analysis System (PSAS)* GEOS-5 Atmospheric General Circulation Model *replaces NSIPP Atmospheric General Circulation Model*
NCEP	Gridpoint Statistical Interpolation (GSI) System (joint with GMAO) *replaces Spectral Statistical Interpolation (SSI)* Global Spectral Forecasting Model WRF regional atmospheric model at 22km resolution CONUS forecast
GMAO	ODAS with OI analysis system with ~10K observations/day
MIT	MITgcm century / millennium adjoint sensitivity

ESMF Adoption Legend

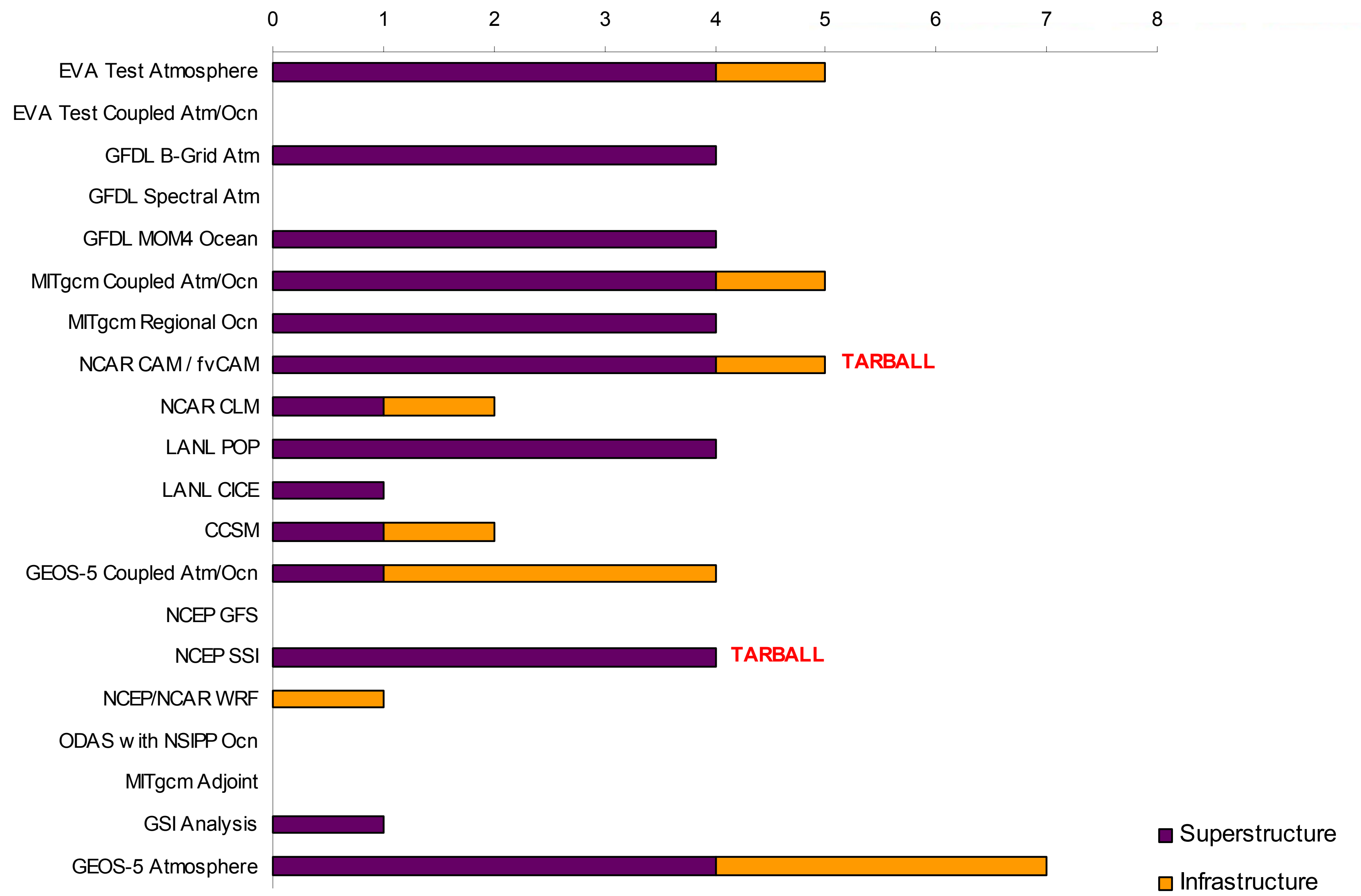
Infrastructure (i = 1…6)

Number indicates how many ESMF utilities are being used internal to the code.

Superstructure (i=1…8)

- Base version of code selected, and configuration decided on (includes version, target platform, validation criteria).
- User component is restructured in an ESMF manner, but may not use ESMF software.
- User component builds valid states and presents standard ESMF interfaces.
- All gridded components run as ESMF stand-alone components - complete for non-coupled applications.
- A system with all components and stub coupler(s) links and runs, even though the coupler may not do anything, or may not use ESMF regridding.
- One field is transferred in some manner through the coupled system.
- ESMF regridding is used if needed.
- All active fields are correctly transferred, and experiment is verified by outside source.

ESMF Adoption Status



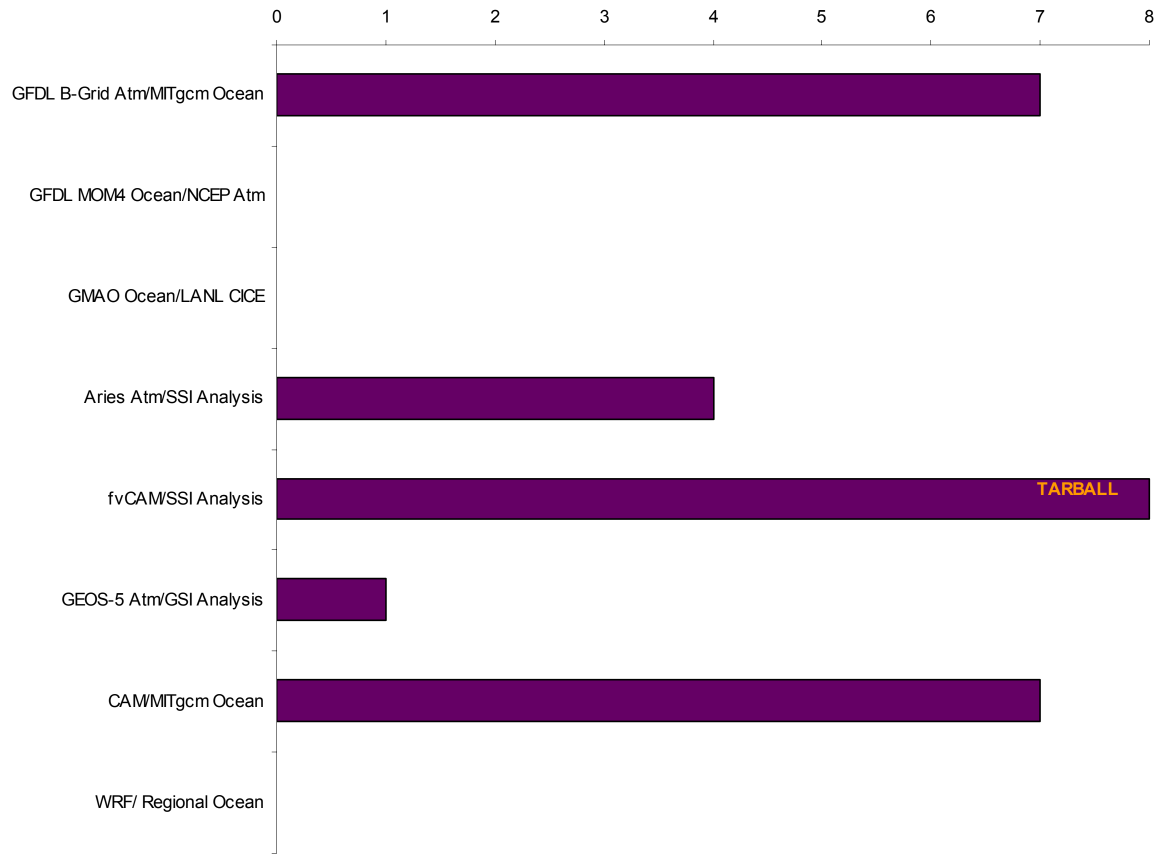
NASA CAN Interoperability Demonstrations

COUPLED CONFIGURATION	NEW SCIENCE ENABLED
GFDL B-grid atm / MITgcm ocn	Global biogeochemistry (CO2, O2), SI timescales.
GFDL MOM4 / NCEP forecast	NCEP seasonal forecasting system.
GMAO ocean / LANL CICE	Sea ice model for extension of SI system to centennial time scales.
NSIPP atm / NCEP analysis	Assimilated initial state for SI.
GMAO GEOS-5/ NCEP GSI	Intercomparison of systems for NASA/NOAA joint center for satellite data assimilation.
NCAR fvCAM/ NCEP analysis	Intercomparison of systems for NASA/NOAA joint center for satellite data assimilation.
NCAR CAM / MITgcm ocn	Improved climate predictive capability: climate sensitivity to large component interchange, optimized initial conditions.
NCEP WRF / Ocean Model	Development of hurricane prediction capability.

NASA CAN Interoperability Experiment Legend

1. Base version of both codes in experiment selected, and configuration decided upon (e.g. target platform, one/two way coupling, fields sent, duration).
2. Both codes run standalone as ESMF components, using component constructs but not necessarily creating valid states.
3. Fields that will be in import/export states of both codes match up with each other.
4. Both codes create valid ESMF import/export states, including fields with ESMF grids.
5. Draft coupler is written and full system with codes, stub coupler, and ESMF can be linked and run on target platform.
6. One field is transferred in some manner in one direction through the coupler.
7. ESMF regridding is used if needed.
8. All fields active in the experiment are correctly transferred and the experiment verified by outside source.

NASA CAN Interoperability Experiment Status



2 STATUS OF DEVELOPMENT AND APPLICATIONS

- Development Status and Priorities
- Performance
- NASA CAN ESMF Project Status
- BEI Codes
- Exercises

Select BEI Modeling Codes

SOURCE	APPLICATION
Navy	Hybrid Coordinate Ocean Model (HYCOM) Navy Coastal Ocean Model (NCOM) Navy Layered Ocean Model (NLOM) Coupled Ocean Atmosphere Mesoscale Prediction System (COAMPS) Global and regional Wave Model (WAM) Advanced Circulation coastal and estuarine model (ADCIRC)
Air Force	HAF Kinematic Solar Wind Global Assimilation of Ionospheric Measurements (GAIM)

2 STATUS OF DEVELOPMENT AND APPLICATIONS

- Development Status and Priorities
- Performance
- NASA CAN ESMF Project Status
- BEI Codes
- Exercises

Exercises

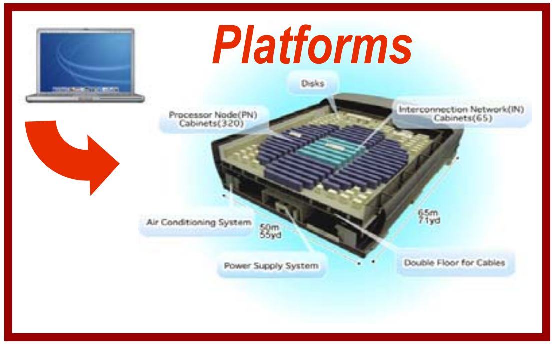
1. Locate on the ESMF website:
   - The Reference Manual, User’s Guide and Developer’s Guide
   - The Project Plan
   - The current task list
   - The modules in the contributions repository
   - The weekly regression test schedule
   - Known bugs from the last public release
   - The % of public interfaces tested
   - The schedule of Early Adopter (Users Group) meetings
   - The ESMF Support Policy

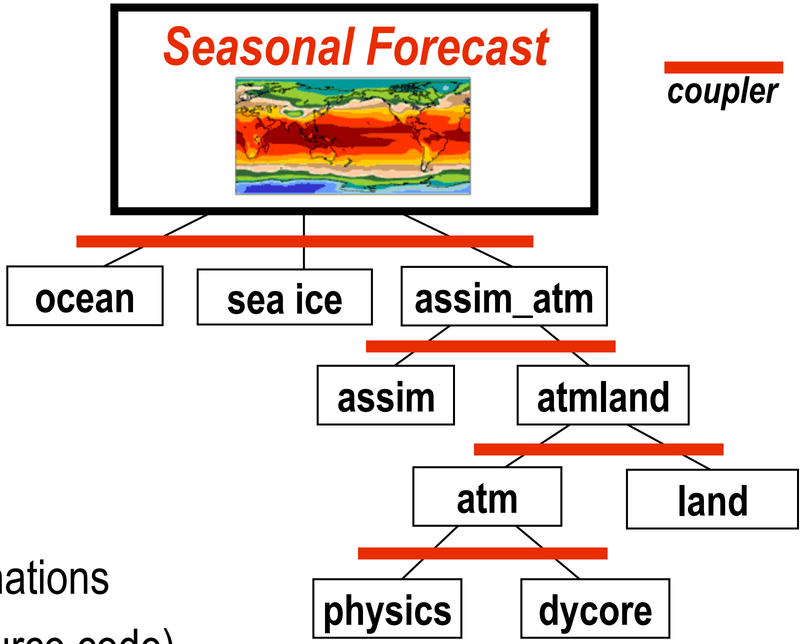
3 DESIGN AND PRINCIPLES OF ESMF

- Computational Characteristics of Weather and Climate
- Design Strategies
- Parallel Computing Definitions
- Framework-Wide Behavior
- Required Methods
- Class Structure
- Exercises

Computational Characteristics of Weather/Climate

- Mix of global transforms and local communications
- Load balancing for diurnal cycle, event (e.g. storm) tracking
- Applications typically require 10s of GFLOPS, 100s of PEs – but can go to 10s of TFLOPS, 1000s of PEs
- Required Unix/Linux platforms span laptop to Earth Simulator
- Multi-component applications: component hierarchies, ensembles, and exchanges; components in multiple contexts
- Data and grid transformations between components
- Applications may be MPMD/SPMD, concurrent/sequential, combinations
- Parallelization via MPI, OpenMP, shmem, combinations
- Large applications (typically 100,000+ lines of source code)



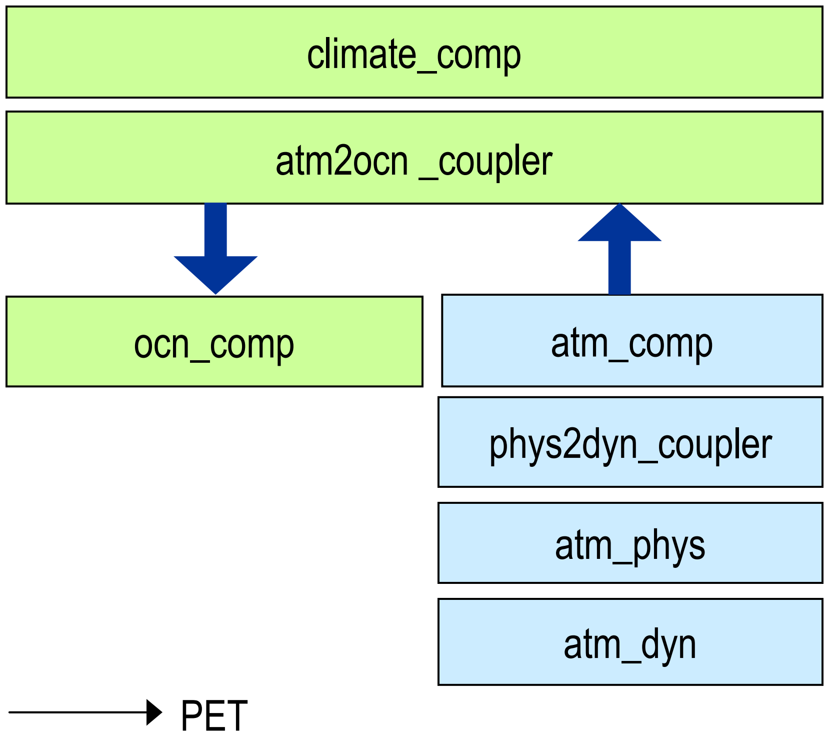


3 DESIGN AND PRINCIPLES OF ESMF

- Computational Characteristics of Weather and Climate
- Design Strategies
- Parallel Computing Definitions
- Framework-Wide Behavior
- Required Methods
- Class Structure
- Exercises

Design Strategy: Intracomponent Communication

All communication in ESMF is handled within components. This allows the architecture of the framework to be independent of the communication strategy. The result is that there is flexibility in implementation of communications and component drivers are straightforward.

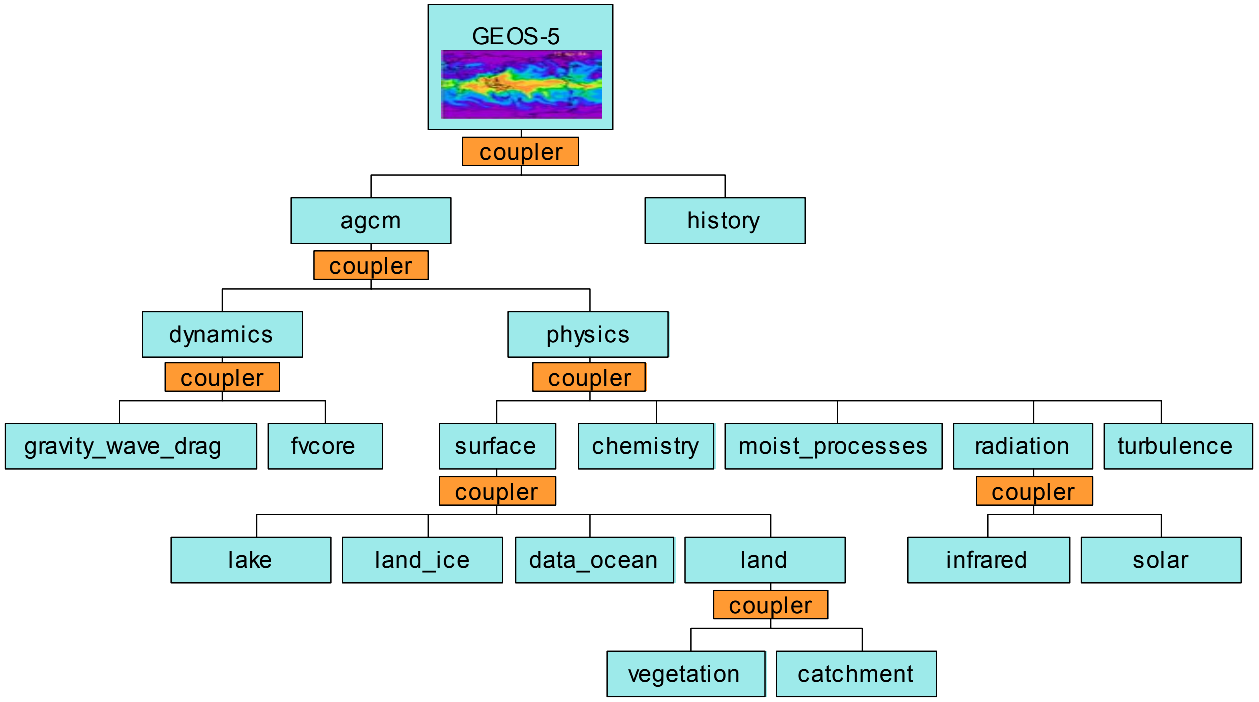


As a consequence, Coupler Components must be defined on the union of the PETs of all the Gridded Components that they couple.

In this example, in order to send data from the atmosphere Component to the ocean, the atm2ocn_coupler mediates the send.

Design Strategy: Hierarchical Applications

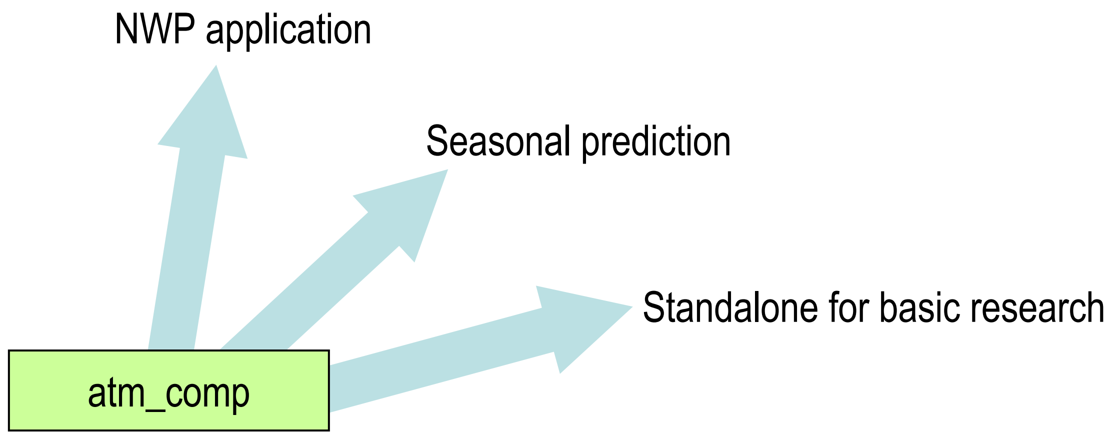
Since each ESMF application is also a Gridded Component, entire ESMF applications can be nested within larger applications. This strategy can be used to systematically compose very large, multi-component codes.



Design Strategy: Modularity

Gridded Components don't have access to the internals of other Gridded Components, and don't store any coupling information. Gridded Components pass their States to other components through their argument list.

Since components are not hard-wired into particular configurations and do not carry coupling information, components can be used more easily in multiple contexts.



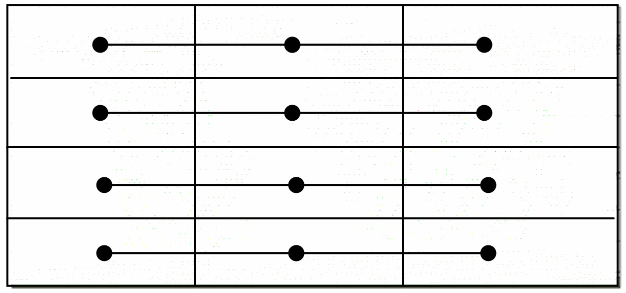
Design Strategy: Uniform Communication API

The same programming interface is used for shared memory, distributed memory, and combinations thereof. This buffers the user from variations and changes in the underlying platforms.

Virtual Machine (VM) = abstraction of machine architecture (num_nodes, num_pes_per_node, etc.)

DE = a decomposition element - may be virtual, thread, MPI process

DELayout = an arrangement of DEs, in which dimensions requiring faster communication may be specified and resources arranged accordingly



4 x 3 DELayout:

The data in a Grid is decomposed according to the number and topology of DEs in the DELayout

3 DESIGN AND PRINCIPLES OF ESMF

- Computational Characteristics of Weather and Climate
- Design Strategies
- Parallel Computing Definitions
- Framework-Wide Behavior
- Required Methods
- Class Structure
- Exercises

Elements of Parallelism

- Decomposition Element (DE)
    - In ESMF a decomposition is represented as a set of Decomposition Elements (DEs).
    - A decomposition that has four pieces in the x direction and three pieces in the y direction would be 4 x 3 DEs
    - A DE is not tied to a particular chunk of data
    - A DE is not tied to a particular processor or other compute resource
    - Sets of DEs are represented by the DELayout class
- Persistent Execution Thread (PET)
    - Path for executing an instruction sequence
    - Sets of PETs are represented by the Virtual Machine (VM) class
- Processing Element (PE)
    - The smallest physical processing unit available on a particular hardware platform

Modes of Parallelism

- Serial vs parallel
  - Serial code runs on one Persistent Execution Thread (PET)
  - Parallel code runs on multiple PETs
- Sequential vs concurrent
  - In sequential mode components run one after the other on the same set of PETs
  - In concurrent mode components run at the same time on different sets of PETs
- SPMD vs MPMD
  - In Single Program Multiple Datastream (SPMD) mode the same program runs across all PETs in the application - components may run sequentially or concurrently.
  - In Multiple Program Multiple Datastream (MPMD) mode the application consists of separate programs launched as separate executables - components may run concurrently or sequentially, but in this mode almost always run concurrently

Local vs Global

- Global means across the whole object or the whole application, depending on the context
- Local must be qualified in ESMF
  - PE local?
  - PET local?
  - DE local?

3 DESIGN AND PRINCIPLES OF ESMF

- Computational Characteristics of Weather and Climate
- Design Strategies
- Parallel Computing Definitions
- Framework-Wide Behavior
- Required Methods
- Class Structure
- Exercises

Framework-Wide Behavior

ESMF has a set of interfaces and behaviors that hold across the entire framework. This consistency helps make the framework easier to learn and understand.

For more information, see Sections 6-8 in the Reference Manual.

Classes and Objects in ESMF

- The ESMF Application Programming Interface (API) is based on the object-oriented programming notion of a class. A class is a software construct that's used for grouping a set of related variables together with the subroutines and functions that operate on them. We use classes in ESMF because they help to organize the code, and often make it easier to maintain and understand.
- A particular instance of a class is called an object. For example, Field is an ESMF class. An actual Field called temperature is an object.

Classes and Fortran

- The Fortran interface is implemented so that the variables associated with a class are stored in a derived type. For example, an `ESMF_Field` derived type stores the data array, grid information, and metadata associated with a physical field.
- The derived type for each class is stored in a Fortran module, and the operations associated with each class are defined as module procedures. We use the Fortran features of generic functions and optional argumentsextensively to simplify our interfaces.

Interface Convention

Methods in ESMF generally look like this:

```
call ESMF_<ClassName><Operation>(classname,
  firstArgument,
secondArgument, ..., rc)
```

where

- `<ClassName>` is the class name,
- `<method>` is the name of the action to be performed,
- `classname` is a variable of the derived type associated with the class,
- the `*arguments` are whatever other variables are required for the operation,
- and `rc` is a return code.

Standard Methods

- `ESMF_<Class>Create()` and `ESMF_<Class>Destroy()`, for allocating and constructing classes and freeing the memory for classes and destructing their internals.
- `ESMF_<Class>Set()` and `ESMF_<Class>Get()`, for setting and retrieving a particular item or flag. In general, these methods are overloaded for all cases where the item can be manipulated as a name/value pair.
- `ESMF_<Class>Add()`, `ESMF_<Class>Get()`, and `ESMF_<Class>Remove()` for manipulating items that can be appended or inserted into a list of like items within a class.
- `ESMF_<Class>Print()`, for printing the contents of a class to standard out. This method is mainly intended for debugging.
- `ESMF_<Class>ReadRestart()` and `ESMF_<Class>WriteRestart()`, for saving the contents of a class and restoring it exactly. These are not yet implemented.
- `ESMF_<Class>Validate()`, for determining whether a class is internally consistent.

Deep and Shallow Classes

- Deep classes require `ESMF_<Class>Create()` and `ESMF_<Class>Destroy()` calls. They take significant time to set up (off the heap) and should not be created in a time-critical portion of code. Deep objects persist even after the method in which they were created has returned. Most classes in the ESMF, including Fields, Bundles, Arrays, Grids and Clocks, fall into this category.
- Shallow classes do not require `ESMF_<Class>Create()` and `ESMF_<Class>Destroy()` calls. They can simply be declared and their values set using an `ESMF_<Class>Set()` call. Shallow classes do not take long to set up (off the stack) and can be declared and set within a time-critical code segment. Shallow objects stop existing when the method in which they were declared has returned. Times and Time Intervals are examples of shallow classes.

3 DESIGN AND PRINCIPLES OF ESMF

- Computational Characteristics of Weather and Climate
- Design Strategies
- Parallel Computing Definitions
- Framework-Wide Behavior
- Required Methods
- Class Structure
- Exercises

Required Calls

- The modules for ESMF are bundled together and can be accessed with a single `USE` statement, `USE ESMF_Mod`.
- `ESMF_Initialize()` and `ESMF_Finalize()` are required methods that must bracket the use of ESMF within an application. They manage the resources required to run ESMF and shut it down gracefully.

Initialize, Run, and Finalize

- `ESMF_<Grid|Cpl>CompInitialize()`, `ESMF_<Grid|Cpl>CompRun()`, and `ESMF_<Grid|Cpl>CompFinalize()` are component methods that are used at the highest level within ESMF. The content of these methods is not part of the ESMF. Instead the methods call into associated Fortran subroutines within user code.

- User components must be segmented into clear initialize, run, and finalize methods that use ESMF prescribed interfaces before they can become ESMF components.

SetServices

- Every ESMF_<Grid|Cpl>Comp is required to provide and document a set services routine.
- The function of the set services subroutine is to register the rest of the required functions in the component, currently initialize, run, and finalize methods. The ESMF method `ESMF_<Grid|Cpl>CompSetEntryPoint()` should be called for each of the required subroutines.
- The AppDriver or parent component code which is creating a component will first call `ESMF_<Grid|Cpl>CompCreate()` to create an "empty" component, and then must call the component-specific set services routine to associate ESMF-standard methods to user-code methods, and to create the VM in which this component will run.
- After set services has been called, the framework now will be able to call the component's initialize, run, and finalize routines as required.

SetServices (cont.)

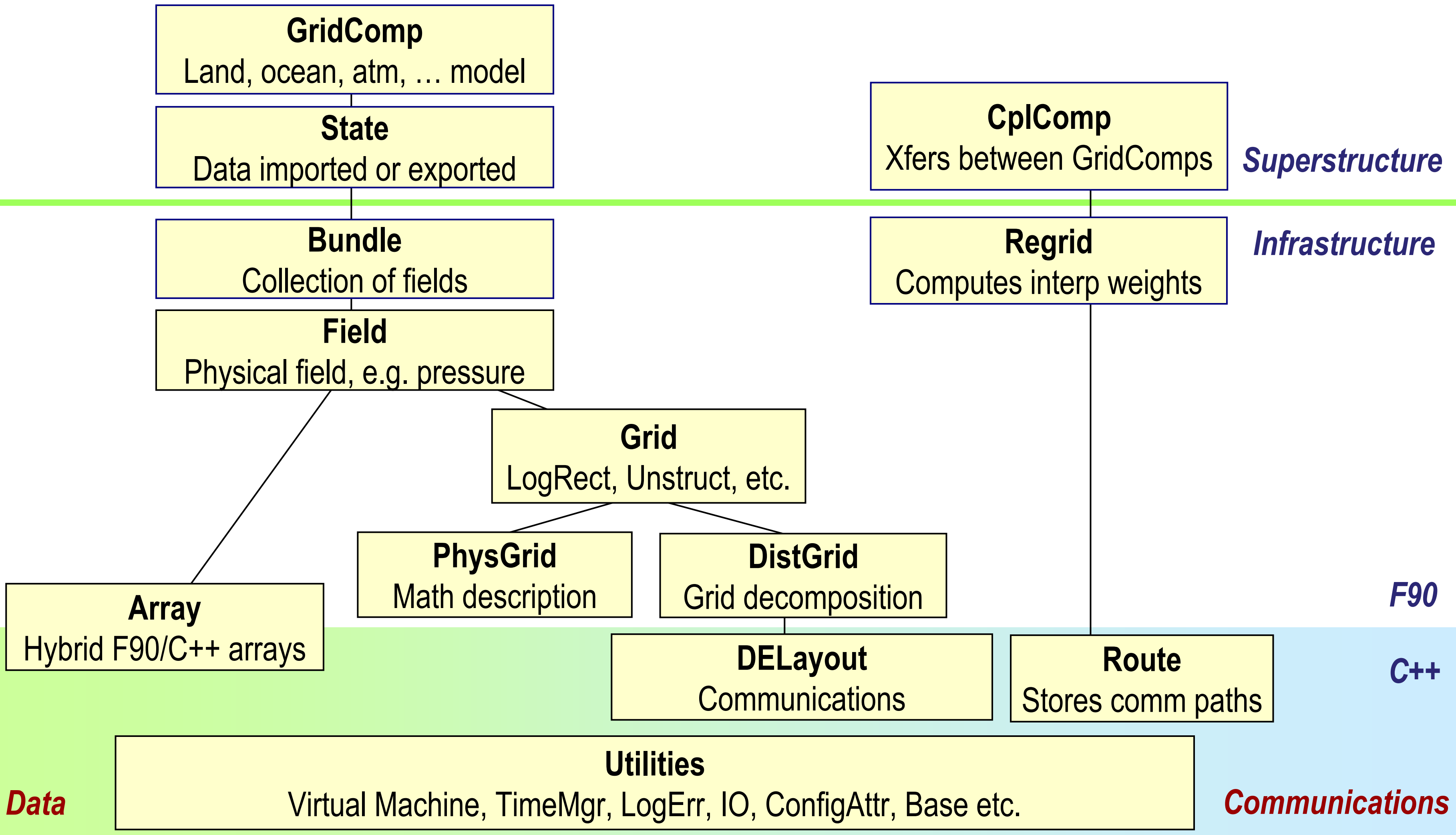
- The set services subroutine name is not predefined (it does not need to be "SetServices" - it is set by the component writer.
- The names of the initialize, run, and finalize user-code subroutines do not need to be public - in fact it is far better for them to be private to lower the chances of public symbol clashes between different components.
- Within the set services routine, the user can also register a private data block by calling the `ESMF_<Grid|Cpl>CompSetInternalState` method.

See Section 14.3 in the Reference Manual for set services examples and 14.6 for `ESMF_GridCompSetServices()` and `ESMF_GridPointSetEntryPoint()` interfaces.

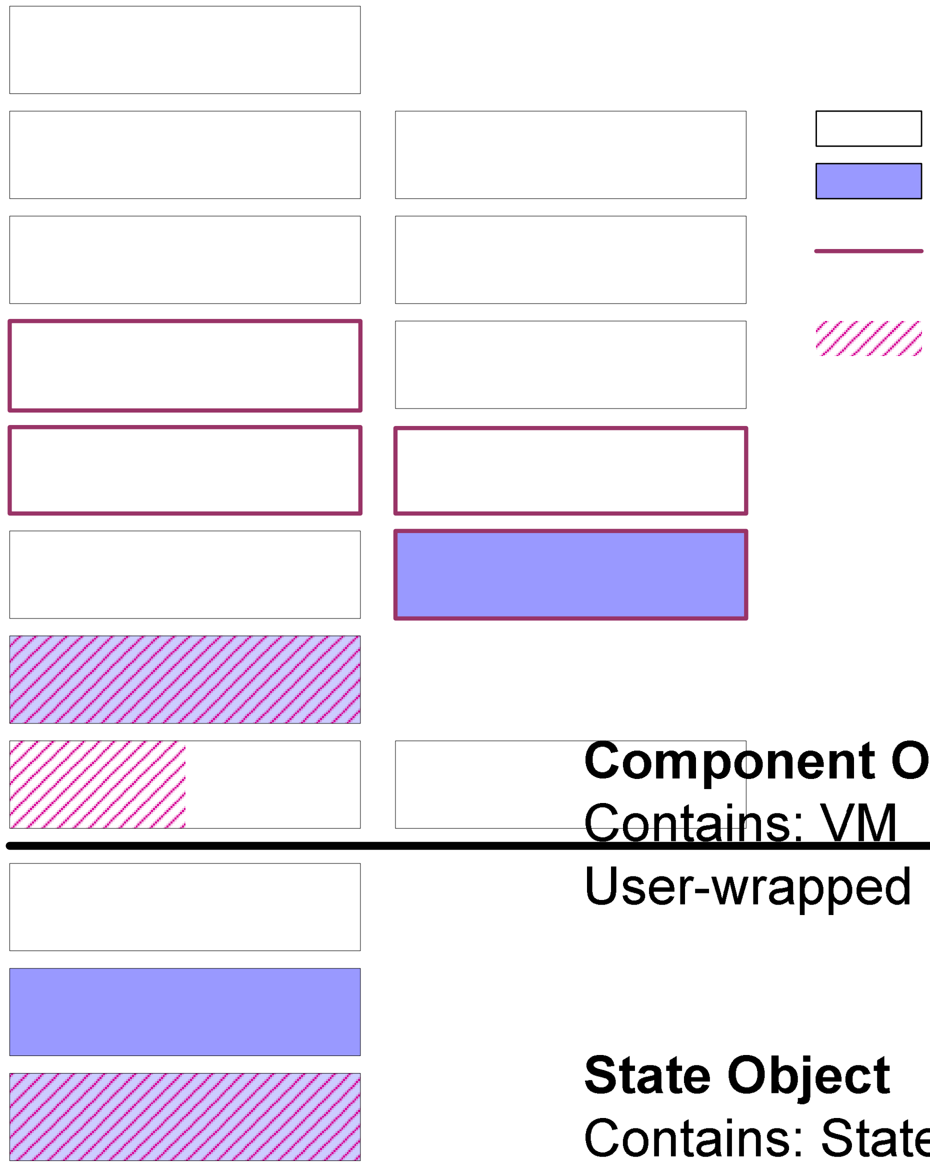
3 DESIGN AND PRINCIPLES OF ESMF

- Computational Characteristics of Weather and Climate
- Design Strategies
- Parallel Computing Definitions
- Framework-Wide Behavior
- Required Methods
- Class Structure
- Exercises

ESMF Class Structure



Current Class Hierarchy

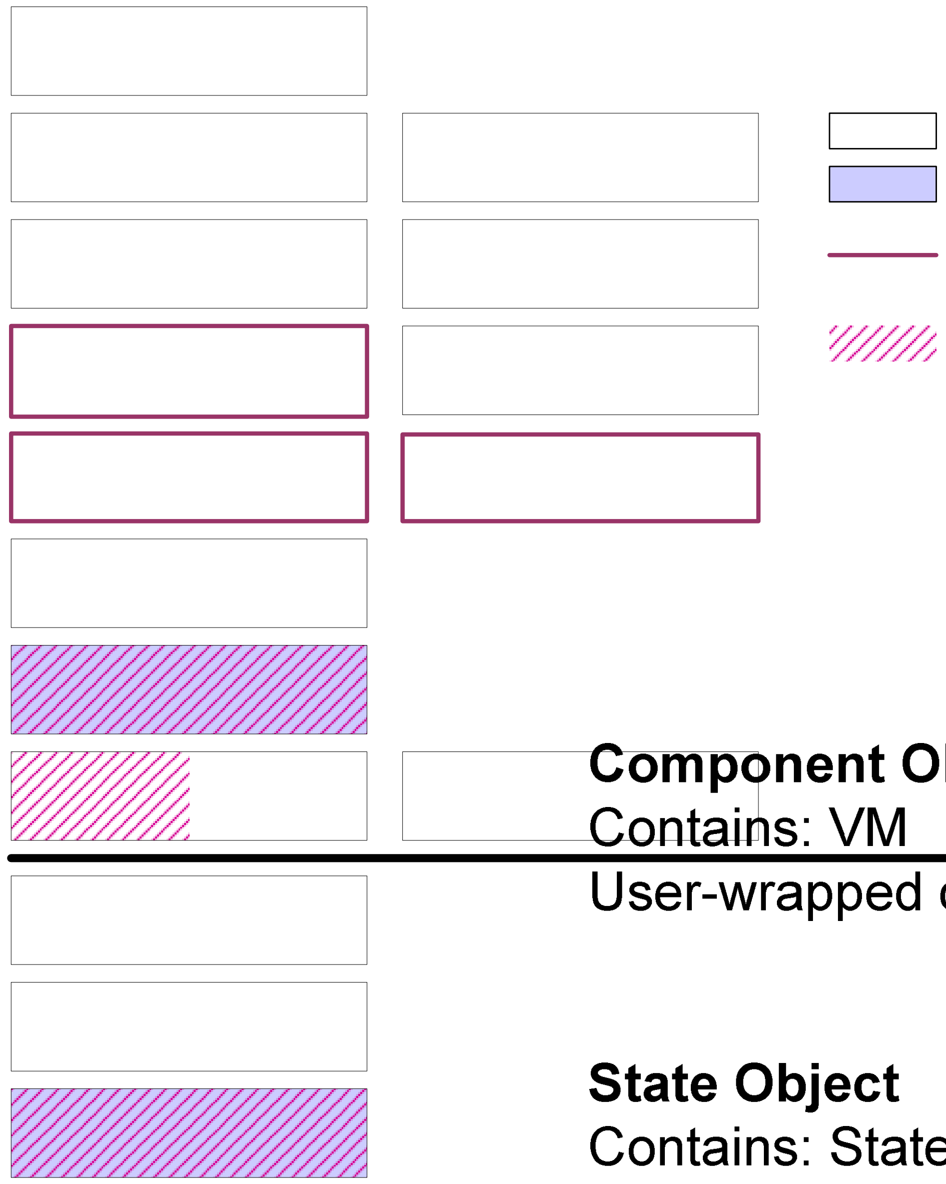


Objects

Component Object
Contains: VM
User-wrapped component structure

State Object
Contains: State, Field, Bundle, Array

Planned Changes



Objects

Component Object
Contains: VM
User-wrapped component structure

State Object
Contains: State, Field, Bundle, Array

3 DESIGN AND PRINCIPLES OF ESMF

- Computational Characteristics of Weather and Climate
- Design Strategies
- Parallel Computing Definitions
- Framework-Wide Behavior
- Required Methods
- Class Structure
- Exercises

Exercises

1. Download Version 2.1.0.
2. Compile.

4 CLASSES AND FUNCTIONS

- ESMF Superstructure Classes
- ESMF Infrastructure Classes: Data Structures
- ESMF Infrastructure Classes: Utilities
- Exercises

ESMF Superstructure Classes

See Sections 12-16 in the Reference Manual.

- Gridded Component
  - Models, data assimilation systems - “real code”
- Coupler Component
  - Data transformations and transfers between Gridded Components
- State – Packages of data sent between Components
- Application Driver – Generic driver

ESMF Components

- An ESMF component has two parts, one that is supplied by the ESMF and one that is supplied by the user. The part that is supplied by the framework is an ESMF derived type that is either a Gridded Component (`GridComp`) or a Coupler Component (`CplComp`).
- A Gridded Component typically represents a physical domain in which data is associated with one or more grids - for example, a sea ice model.
- A Coupler Component arranges and executes data transformations and transfers between one or more Gridded Components.
- Gridded Components and Coupler Components have standard methods, which include initialize, run, and finalize. These methods can be multi-phase.

ESMF Components (cont.)

- The second part of an ESMF component is user code, such as a model or data assimilation system. Users set entry points within their code so that it is callable by the framework. In practice, setting entry points means that within user code there are calls to ESMF methods that associate the name of a Fortran subroutine with a corresponding standard ESMF operation.
- EXAMPLE A user-written initialization routine called popOceanInit might be associated with the standard initialize routine of an ESMF Gridded Component named “POP” that represents an ocean model.

ESMF Gridded Components

- Gridded Components are models, data assimilation systems, diagnostics, etc.
- Gridded Components can be nested
- Gridded Components can be run as ensembles
- Depending on how the current code is structured, may be possible to wrap without structural changes
- Or might use ESMF conversion as a reason to make structural changes!
- States for import/export
- Sequential and concurrent modes of execution possible
- Registration routine (SetServices) to associate user code routines with standard ESMF intialize/run/finalize methods

ESMF Coupler Components

- Coupler Components perform the transformations and transfers between Gridded Components
- States for import/export
- Not automatic - must to be customized for each new configuration
- Expected to be thin, however - making use of the transformation routines in ESMF

ESMF States

- All data passed between Components is in the form of States and States only
- Description/reference to other ESMF data objects
- Data is referenced so does not need to be duplicated
- Can be Bundles, Fields, Arrays, States, or name-placeholders

Application Driver

- Small, generic program that contains the “main” for an ESMF application.

4 CLASSES AND FUNCTIONS

- ESMF Superstructure Classes
- ESMF Infrastructure Classes: Data Structures
- ESMF Infrastructure Classes: Utilities
- Exercises

ESMF Infrastructure Data Classes

Model data is contained in a hierarchy of multi-use classes. The user can reference a Fortran array to an Array or Field, or retrieve a Fortran array out of an Array or Field.

- Array – holds a Fortran array (with other info, such as halo size)
- Field – holds an Array, an associated Grid, and metadata
- Bundle – collection of Fields on the same Grid bundled together for convenience, data locality, latency reduction during communications

Supporting these data classes is the Grid class, which represents a numerical grid

Grids

See Section 25 in the Reference Manual for interfaces and examples.

- The ESMF Grid class represents all aspects of the computational domain and its decomposition in a parallel-processing environment It has methods to internally generate a variety of simple grids
- The ability to read in more complicated grids provided by a user is not yet implemented
- ESMF Grids are currently assumed to be two-dimensional, logically-rectangular horizontal grids, with an optional vertical grid whose coordinates are independent of those of the horizontal grid
- Each Grid is assigned a staggering in its create method call, which helps define the Grid according to typical Arakawa nomenclature.

Arrays

See Section 22 in the Reference Manual for interfaces and examples.

- The Array class represents a multidimensional array.
- An Array can be real, integer, or logical, and can possess up to seven dimensions. The Array can be strided.
- The first dimension specified is always the one which varies fastest in linearized memory.
- Arrays can be created, destroyed, copied, and indexed. Communication methods, such as redistribution and halo, are also defined.

Fields

See Section 20 in the Reference Manual for interfaces and examples.

- A Field represents a scalar physical field, such as temperature.
- ESMF does not currently support vector fields, so the components of a vector field must be stored as separate Field objects.
- The ESMF Field class contains the discretized field data, a reference to its associated grid, and metadata.
- The Field class provides methods for initialization, setting and retrieving data values, I/O, general data redistribution and regridding, standard communication methods such as gather and scatter, and manipulation of attributes.

Bundles

See Section 18 in the Reference Manual for interfaces and examples.

- The Bundle class represents “bundles” of Fields that are discretized on the same Grid and distributed in the same manner.
- Fields within a Bundle may be located at different locations relative to the vertices of their common Grid.
- The Fields in a Bundle may be of different dimensions, as long as the Grid dimensions that are distributed are the same.
- In the future Bundles will serve as a mechanism for performance optimization. ESMF will take advantage of the similarities of the Fields within a Bundle in order to implement collective communication, IO, and regridding.

ESMF Communications

See Section 27 in the Reference Manual for a summary of communications methods.

- Halo
  - Updates edge data for consistency between partitions
- Redistribution
  - No interpolation, only changes how the data is decomposed
- Regrid
  - Based on SCRIP package from from Los Alamos
  - Methods include bilinear, conservative
- Bundle, Field, Array-level interfaces

ESMF DataMap Classes

These classes give the user a systematic way of expressing interleaving and memory layout, also hierarchically (partially implemented, rework expected)

- ArrayDataMap – relation of array to decomposition and grid, row / column major order, complex type interleave
- FieldDataMap – interleave of vector components
- BundleDataMap – interleave of Fields in a Bundle

4 CLASSES AND FUNCTIONS

- ESMF Superstructure Classes
- ESMF Infrastructure Classes: Data Structures
- ESMF Infrastructure Classes: Utilities
- Exercises

ESMF Utilities

- Time Manager
- Configuration Attributes (replaces namelists)
- Message logging
- Communication libraries
- Regridding library (parallelized, on-line SCRIP)
- IO (barely implemented)
- Performance profiling (not implemented yet, may simply use Tau)

Time Manager

See Sections 32-37 in the Reference Manual for more information.

Time manager classes are:

- Calendar
- Clock
- Time
- Time Interval
- Alarm

These can be used independent of other classes in ESMF.

Calendar

A Calendar can be used to keep track of the date as an ESMF Gridded Component advances in time. Standard calendars (such as Gregorian and 360-day) and user-specified calendars are supported. Calendars can be queried for quantities such as seconds per day, days per month, and days per year.

Supported calendars are:

- **Gregorian** The standard Gregorian calendar, proleptic to 3/1/-4800.
- **no-leap** The Gregorian calendar with no leap years.
- **Julian Day** A Julian days calendar.
- **360-day** A 30-day-per-month, 12-month-per-year calendar.
- **no calendar** Tracks only elapsed model time in seconds.

Clock and Alarm

Clocks collect the parameters and methods used for model time advancement into a convenient package. A Clock can be queried for quantities such as start time, stop time, current time, and time step. Clock methods include incrementing the current time, and determining if it is time to stop.

Alarms identify unique or periodic events by “ringing” - returning a true value - at specified times. For example, an Alarm might be set to ring on the day of the year when leaves start falling from the trees in a climate model.

Time and Time Interval

A Time represents a time instant in a particular calendar, such as November 28, 1964, at 7:31pm EST in the Gregorian calendar. The Time class can be used to represent the start and stop time of a time integration.

Time Intervals represent a period of time, such as 300 milliseconds. Time steps can be represented using Time Intervals.

Clock Creation and Timestepping

See Section 36.2 in the Reference Manual for examples and interfaces.

Config Attributes

See Section 38 in the Reference Manual for interfaces and examples.

- ESMF Configuration Management is based on NASA DAO’s Inpak 90 package, a Fortran 90 collection of routines/functions for accessing *Resource Files* in ASCII format.
- The package is optimized for minimizing formatted I/O, performing all of its string operations in memory using Fortran intrinsic functions.

LogErr

See Section 39 in the Reference Manual for interfaces and examples.

- The Log class consists of a variety of methods for writing error, warning, and informational messages to files.
- A default Log is created at ESMF initialization. Other Logs can be created later in the code by the user.
- A set of standard return codes and associated messages are provided for error handling.
- LogErr will automatically put timestamps and PET numbers into the Log.

LogErr Options

- Buffering allows for writing to a file immediately or storing entries in a buffer. The buffer will either write when full, or when the user calls an `ESMF_LogFlush()` method.
- The user has the capability to halt the program on an error or on a warning by using the `ESMF_LogSet()` method with the halt property
  - `ESMF_LOG_HALTWARNING` - the program will stop on any and all warnings or errors
  - `ESMF_LOG_HALTERROR` - the program will only halt on errors
  - `ESMF_LOG_HALTNEVER` – the program will run through errors
- Single or multi file (per PET) option for writing messages

Virtual Machine (VM)

See Section 41 in the Reference Manual for VM interfaces and examples.

- VM handles resource allocation
- Elements are Persistent Execution Threads or PETs
- PETs reflect the physical computer, and are one-to-one with Posix threads or MPI processes
- Parent Components assign PETs to child Components
- The VM communications layer does simple MPI-like communications between PETs (alternative communication mechanisms are layered underneath)

DELayout

- See Section 40 in the Reference Manual for interfaces and examples.

- Handles decomposition
- Elements are Decomposition Elements, or DEs (decomposition that's 2 pieces in x by 4 pieces in y is a 2 by 4 DELayout)
- DELayout maps DEs to PETs, can have more than one DE per PET (for cache blocking, user-managed OpenMP threading)
- Simple connectivity or more complex connectivity, with weights between DEs - users specify dimensions where greater connection speed is needed
- Array, Field, and Bundle methods perform inter-DE communications

4 CLASSES AND FUNCTIONS

- ESMF Superstructure Classes
- ESMF Infrastructure Classes: Data Structures
- ESMF Infrastructure Classes: Utilities
- Exercises

Exercises

1. Go to the ESMF main source repository via the website (from **Development**).
2. Select **Browse the CVS Tree.**
3. Change directory to **esmf**, which is the ESMF distribution.
4. Change directory to **build**, to view directories for supported platforms.
5. Return to the next level up by clicking on [cvs]/esmf/**esmf**/build.
6. Change directory to **src** and locate the **Infrastructure** and **Superstructure** directories.
7. Note that code is arranged by class within these directories, and that each class has a standard set of subdirectories (doc, examples, include, interface, src, and tests, plus a makefile). This way of browsing the ESMF source code shows all directories, even empty ones.

5 PREPARING FOR AND USING ESMF

- Adoption Strategies
- Exercises

Adoption Strategies: Top Down

1. Decide how to organize the application as discrete Gridded and Coupler Components. The developer might need to reorganize code so that individual components are cleanly separated and their interactions consist of a minimal number of data exchanges.
2. Divide the code for each component into initialize, run, and finalize methods. These methods can be multi-phase, e.g., init_1, init_2.
3. Pack any data that will be transferred between components into ESMF Import and Export State data structures.
4. The user must describe the distribution of grids over resources on a parallel computer via the VM and DELayout.
5. Pack time information into ESMF time management data structures.
6. Using code templates provided in the ESMF distribution, create ESMF Gridded and Coupler Components to represent each component in the user code.
7. Write a set services routine that sets ESMF entry points for each user component's initialize, run, and finalize methods.
8. Run the application using an ESMF Application Driver.

Adoption Strategies: Bottom Up

Adoption of infrastructure utilities and data structures can follow many different paths. The calendar management utility is a popular place to start, since there is enough functionality in the ESMF time manager to merit the effort required to integrate it into codes and bundle it with an application.

ESMF Quickstart

Directory with the shell of an application

- 2 Gridded Components
- 1 Coupler Component
- 1 top-level Gridded Component
- 1 AppDriver main program

5 PREPARING FOR AND USING ESMF

- Adoption Strategies
- Exercises

Exercises

Following the User's Guide:

- Review and run Quickstart.
- Run unit tests and system tests.
- Run examples.

6 RESOURCES

- Documentation
- User Support
- Testing and Validation Pages
- Mailing Lists
- Users Meetings
- Exercises

Documentation

- Users Guide
  - Installation, quick start and demo, architectural overview, glossary
- Reference Manual
  - Overall framework rules and behavior
  - Method interfaces, usage, examples, and restrictions
  - Design and implementation notes
- Developers Guide
  - Documentation and code conventions
  - Definition of compliance
- Requirements Document
- Implementation Report
  - C++/Fortran interoperation strategy
- (Draft) Project Plan
  - Goals, organizational structure, activities

Documentation

- Latex and html documents are automatically generated from code and comments in the ESMF source code using the PROTEX tool from NASA
- The Reference Manual, Users Guide and Requirements Document are archived with the source code in the main ESMF CVS repository and bundled with each release
- These documents can be built by the user with the make utility (latex, latex2html, and dvipdf are needed)
- Code examples from the documentation, quick start, and demo are regression tested nightly and automatically updated

User Support

- ALL requests go through the esmf_support@ucar.edu list so that they can be archived and tracked
- Support policy is on the ESMF website
- Support archives and bug reports are on the ESMF website - see http://www.esmf.ucar.edu > **Development**

  Bug reports are under **Bugs** and support requests are under **Lists**.

Testing and Validation Pages

- Accessible from the **Development** link on the ESMF website
- Detailed explanations of system tests
- Supported platforms and information about each
- Links to regression test archives
- Weekly regression test schedule

Mailing Lists To Join

- esmf_jst@ucar.edu

  Joint specification team discussion
  - Release and review notices
  - Technical discussion
  - Coordination and planning
- esmf_info@ucar.edu

  General information
  - Quarterly updates
- esmf_community@ucar.edu

  Community announcements
  - Annual meeting announcements

Mailing Lists To Write

- esmf

  Project leads

  - Non-technical questions
  - Project information

- esmf_support

  Technical questions and comments

Users Meetings

- Every six weeks ESMF Early Adopters meet at GFDL
- Meeting schedule is on the ESMF website

  http://www.esmf.ucar.edu > **Community**

6 RESOURCES

- Documentation
- User Support
- Testing and Validation Pages
- Mailing Lists
- Users Meetings
- Exercises

Exercises

1. Subscribe to mailing lists.

7 COMPLIANCE

- Definitions
- Exercises

Compliance

- Compliance and adoption are used interchangeably in documents
- Definitions result from negotiation with NASA program staff and apply to the ESMF / NASA contractual obligation
- Partial compliance and full compliance defined
- Partial compliance means use of the superstructure layer
    - User code structured as discrete components with initialize, run, and finalize methods
    - Data to be transferred between components packaged as ESMF States
    - User code wrapped as ESMF Gridded and Coupled Components
    - Application sequenced using ESMF Application Driver
- Full compliance means partial compliance plus using three or more utilities
- Full compliance is not appropriate for all codes

Partial Compliance

In order to achieve partial compliance, a JMC code component must implement, or adopt default implementations, of the complete set of standard ESMF component interface methods including the following capabilities:

- It must be able to be instantiated in parallel configurations.
- It must provide implementations of methods for creation, deletion, configuration, initialization, finalization, run, read and write restart, and others as necessary for control by an ESMF application framework.
- It must provide method implementations to allow it to be queried for its distribution, state (i.e. fields available for export, fields required for import, etc.), run status and other pertinent information.
- Communication with other JMC code components must be mediated by an ESMF coupler component using framework communication services, such that neither JMC component needs to maintain information about the specific component that it is being coupled to.

Partial Compliance (cont.)

- Data and information to be exchanged with other JMC code components must be provided through ESMF constructs and utilities (i.e. ESMF state, bundles, elds, time, grid, decomposition, etc.) These must include pertinent metadata information and provide a standard format for exchanging information. JMC code components must use the public interface methods provided by the ESMF utilities and constructs and not directly manipulate their internal data.
- The JMC components must be able to accept ESMF time management information.
- Data and information to be exchanged with other JMC code components must be provided through ESMF constructs and utilities (i.e. ESMF state, bundles, elds, grid, etc.) These must include pertinent metadata information and provide a standard format for exchanging information. JMC code components must use the public interface methods provided by the ESMF utilities and not directly manipulate the internal data of those utilities.

Full Compliance

A fully compliant JMC code component must satisfy all requirements described for partial compliance. In addition, a fully compliant component must:

- Extensively use internally three or more utilities from the following set: I/O, parameter specification, log/error, performance profiling, time management, grid communication services.
- Adopt the standard ESMF grid communication services and constructs internally to the extent necessary to allow interoperability with other compliant weather, climate, and data assimilation components.
- Adopt design features that eliminate or minimize as much as possible the potential for name space conflicts of variables, methods, etc. between components.
- Adopt design features that eliminate or minimize as much as possible the potential for I/O conflicts between components during reads/writes of configuration, state, errors, logs, performance analysis, etc.

7 COMPLIANCE

- Definitions
- Exercises

Exercises

1. Consider what level of compliance is appropriate for your application.
2. Consider whether you would use a top-down or bottom-up strategy for adoption.

8 CODE EXAMPLES

- Users can discuss adoption of ESMF in their applications with ESMF staff.